A MARSHALL CAVENDISH **17** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol. 2 **No 17**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Dave King. Pages 509, 510, Projection Audio Visual. Pages 514, 516, 517, 519, Paddy Mounter. Pages 520, 523, 524, 527, Dave King. Pages 528, 530, Jeremy Gower. Pages 529, 531, Ray Duns. Pages 532, 533, Chris Lyon. Pages 534, 536, 538, Gary Wing.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN
Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015
New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington
Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).
UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN
Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

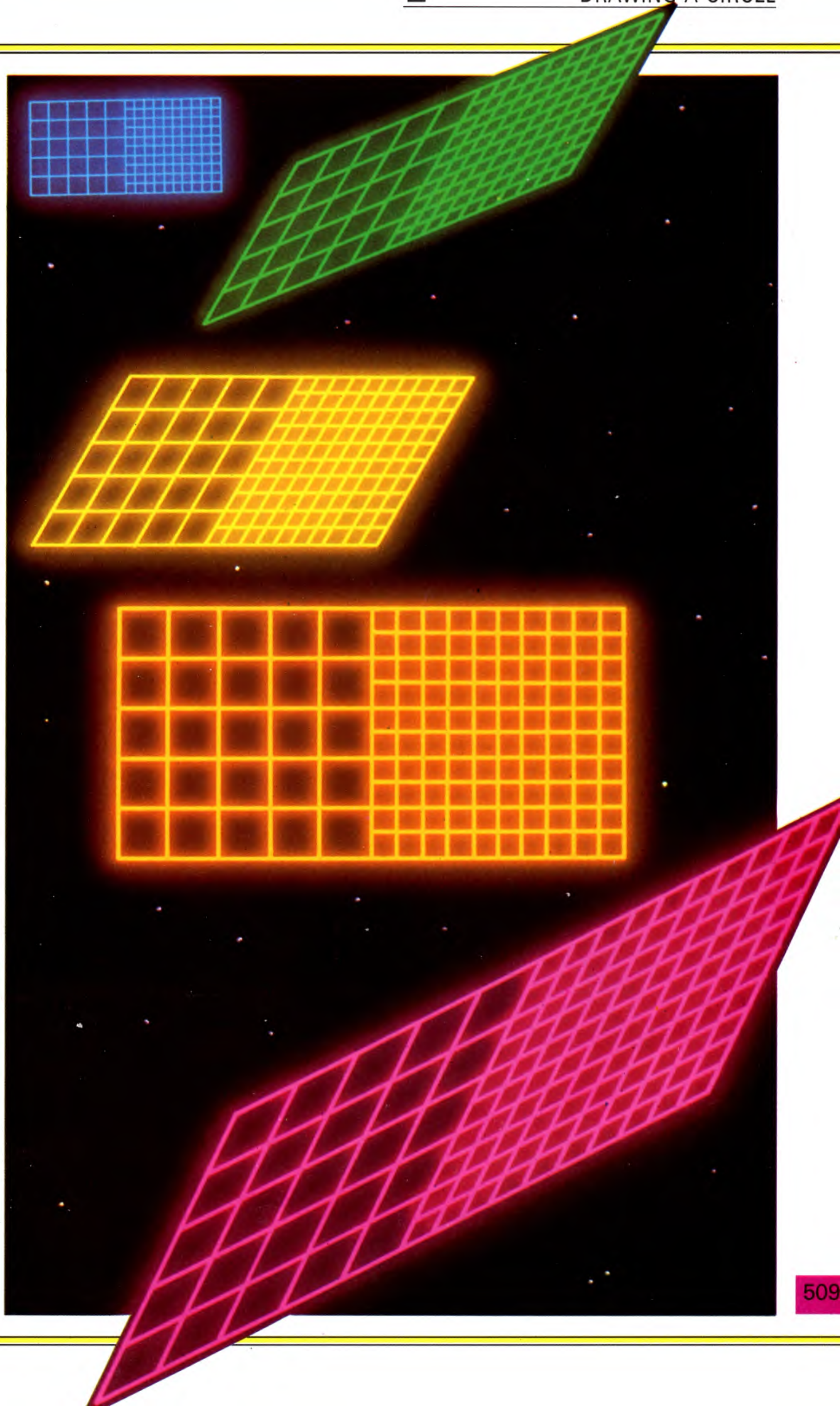
The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

IT'S A FRAME-UP

In the first of a series of articles on wireframe drawing, find out how to construct grids and circles. Later, you'll see how these become the building blocks of 3-D images.

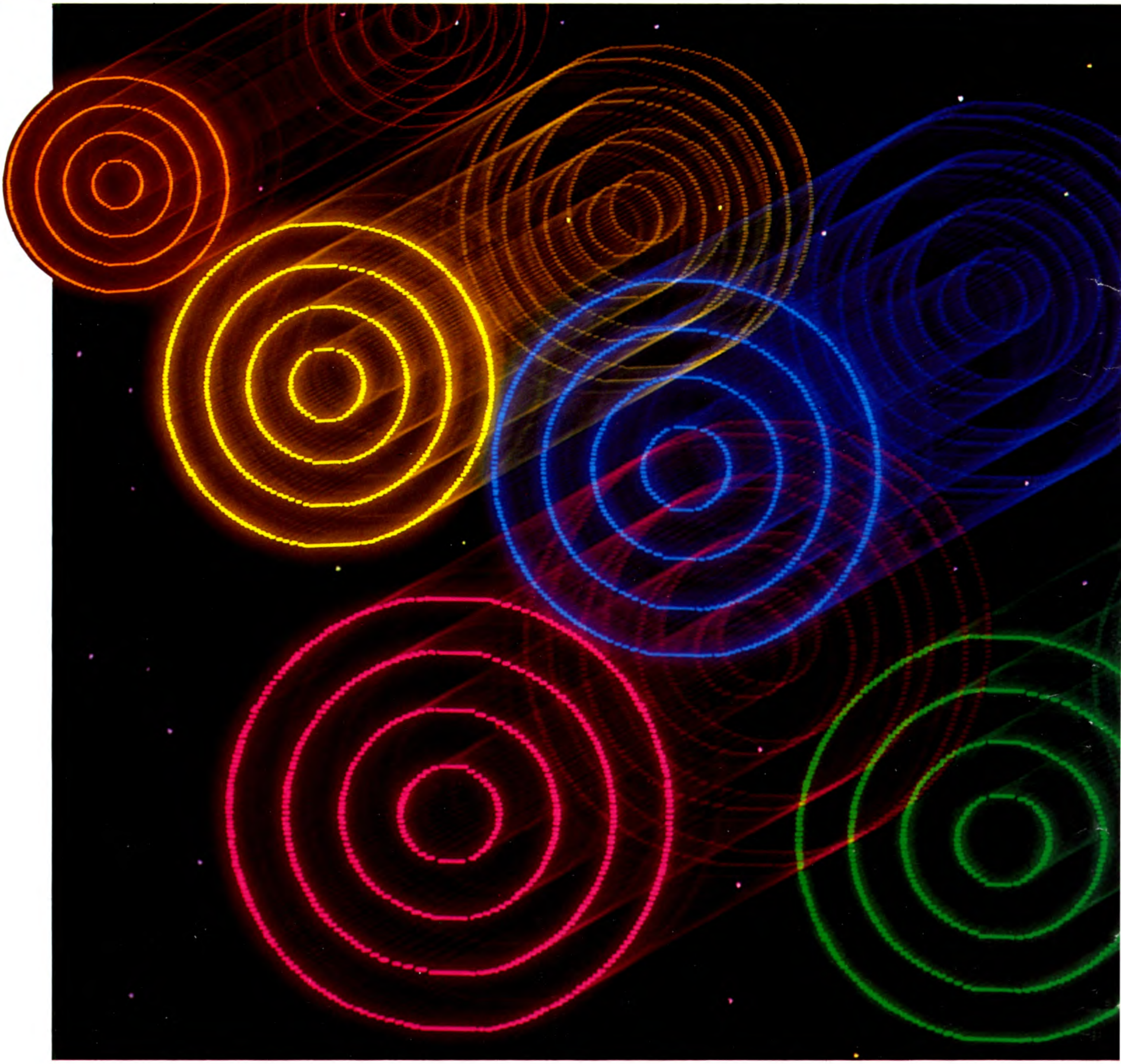


Animated engineering drawings of cars and other machinery have become popular advertising images—and most impressive they are, too. These 'wireframe drawings' are among the latest design aids made possible by computer graphics. If, as a home computer user, you have longed to produce such animated images on your TV screen, then you will be disappointed to learn that your micro cannot match the splendour of advertising images produced jointly by computers, artists and photography experts. All is not lost, however, because you can learn in this series of articles how to draw and manipulate three-dimensional images in a way that many design engineers will envy.

Your computer's ability to address individual points or pixels (picture elements) on the screen provides a powerful means of drawing as you have seen in a number of graphics articles. For example, the Spectrum has 256 pixels horizontally and 176 pixels vertically. Normally you can plot straight lines or points. Combined with colour facilities, these can be used to provide high-resolution displays.

The normal method of drawing lines on computer controlled graphics systems is similar to the way you use a pen and paper. You can move the cursor over the screen without marking it, or you can move it on the screen to leave a mark. Changing colour is as simple as changing pens. The main difference between the computer system and a person using pen and paper is that the computer is faster and better at drawing straight lines—a task that is difficult for people, without the use of a ruler.

This line drawing facility allows you to draw outline images, and also 'wireframe' pictures of three-dimensional objects. Wireframe pictures are an imaginary representation which consist of a grid of lines over the surface of the object. Often no attempt is made to hide the lines at the back of the object—to do so

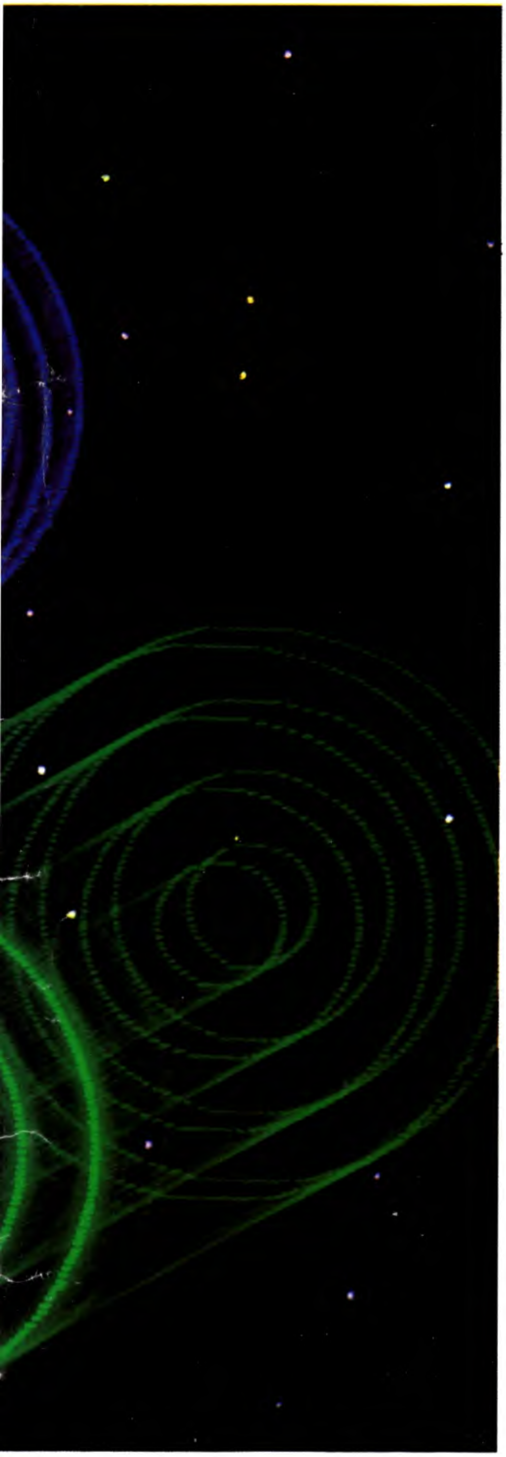


requires a considerable amount of extra computation. The result looks as if the object is made from a wire frame joined to form a grid or mesh.

This type of display can be animated by turning and moving the wireframe object, as in the advertising images of a car, say. The rapid animation is not practicable on a home computer, because each frame needs to be displayed at a rate of at least 25 per second for smooth animation, and a home computer takes much longer than $\frac{1}{25}$ of a second for all but the simplest displays. Indeed, most commercial computer animated displays are not generated in 'real time' (as they happen on the screen). Instead, each picture frame is generated separately, taking seconds, minutes or even hours for complicated high-resolution displays, and then saved on film or video tape for viewing later at the correct speed.

For static displays, time is not important, although you might become impatient waiting for the picture to be completed. Often, however, it is just as interesting, if not more so, to watch a wireframe picture being built up on the screen, as it is to see the finished object—especially with some of the more complicated drawings.

The drawing routines in this article are beyond the scope of the ZX81. Users of the



Commodore 64 need a Simons' BASIC cartridge, and the Vic 20 needs a Super Expander cartridge in order to RUN these programs.

It is best to begin by drawing simple shapes—such as a cube or sphere—then as you gain experience, you can try drawing more complicated and interesting shapes. In this article you will learn how to generate some two-dimensional shapes and to make them appear to be in three-dimensional space.

BASIC STEPS

Even with the high-speed capacity of your computer, and its ability to draw lines, it takes a great deal of work to produce a complicated wireframe picture. The basic steps are in a set of routines, which include the basic drawing commands. Essentially, you need a command to move the cursor without drawing, and one to draw from the last cursor position to a new one. These commands are slightly different from machine to machine. They are the PLOT, MOVE, DRAW and LINE commands you have used often in the other graphics articles.

At the start of a graphics program you normally set the computer to a graphics mode, if necessary, and clear the screen. If you have a choice of screen resolution, choose the highest. It is also a good idea to structure the program, so that all the drawing commands are collected together into subroutines. Besides being sound programming practice, this methodical approach prevents you frequently having to rewrite sections of program that achieve the same results. And because all the commands that perform a certain task are collected in one section of program, it is a simple matter to develop or expand the program and to understand it. Structuring actually slows down the program, but this is not important for static displays, and the greater flexibility it gives—particularly for an application such as wireframe drawing—is a great advantage.

INITIALIZING THE MACHINE

To set up the first stage of these drawing routines, enter this section of programming, but do not RUN it yet, because it is incomplete.

```
9000 REM INIT
9010 BORDER 4: PAPER 7: INK 0: CLS:LET
   N=0
9070 RETURN
9100 REM MOVE
9110 PLOT X,Y
9120 RETURN
9200 REM DRAW
9210 DRAW X—PEEK 23677,Y—PEEK 23678
9220 RETURN
```

```
9000 PRINT "♥"
9030 RETURN
```

```
9000 SCNCLR
9030 RETURN
```

```
9000 DEF PROCINIT
9010 CLS:CLG
9030 ENDPROC
9100 DEF PROCMOVE(X, Y)
9110 MOVE X, Y
9120 ENDPROC
9200 DEF PROCDRAW(X, Y)
9210 DRAW X, Y
9220 ENDPROC
```

```
9000 PCLS
9030 RETURN
```

The Commodore 64, Vic 20, Dragon and Tandy programs are shorter because these computers allow you both to move the cursor *and* draw on the screen using a single command. The Spectrum and Acorn programs set up subroutines to move the cursor position without drawing (Lines 9100 to 9120) and to draw on the screen (Lines 9200 to 9220). This is so that these routines can be combined to make a single moving and drawing routine.

Now, when you wish to draw a shape, you need not tell the computer how to move the cursor or how to mark the screen to form each different shape. Instead, it is much simpler to define each new shape in terms of these basic routines.

LINE DRAWING

The simplest shape you could wish to draw is a line, so here is a routine to do it (you still won't be able to RUN the program):

```
9500 REM LINE
9510 LET X=XS: LET Y=YS: GOSUB 9100
9520 LET X=XE: LET Y=YE: GOSUB 9200
9550 RETURN
```

```
9500 LINE XS,YS,XE,YE,1
9550 RETURN
```

```
9500 DRAW 1,XS,YS TO XE,YE
9550 RETURN
```

```
9500 DEF PROCLINE(XS,YS,XE,YE)
9510 PROCMOVE(XS,YS)
9540 PROCDRAW(XE,YE)
9550 ENDPROC
```

```
9500 LINE(XS,YS)—(XE,YE),PSET
9550 RETURN
```

This routine specifies a start position, coordinates (XS,YS), and an end position, coordinates

(XE,YE), for the line. Some computers need DRAW or LINE commands; for others these are already specified in the routines from Line 91ØØ to Line 922Ø. The line-drawing routine is the basis of most wireframe drawing programs, so it can be used to establish one of the 'building blocks' in the drawing process—a grid.

DRAWING A GRID

To depict a surface and any irregularities, such as cracks, hills and dales it might contain, it is best to visualize it not as one continuous area enclosed in a rectangle, but as a grid of horizontal and vertical lines. Any irregular features in the surface can be shown as distortions of these lines. Enter the next section of program (but do not RUN) to define a routine to draw a grid:

```
5ØØØ LET JA=LW/NX
5Ø1Ø LET XS=XA
5Ø2Ø FOR J=Ø TO NX
5Ø25 LET YS=YA: LET XE=XS: LET
   YE=YA+LH
5Ø3Ø GOSUB 95ØØ
5Ø4Ø LET XS=XS+JA
5Ø5Ø NEXT J
5Ø6Ø LET JA=LH/NY
5Ø7Ø LET YS=YA+LH
5Ø8Ø FOR J=Ø TO NY
5Ø9Ø LET XS=XA+LW: LET XE=XA: LET
   YE=YS
51ØØ GOSUB 95ØØ
511Ø LET YS=YS-JA
512Ø NEXT J
513Ø RETURN
```

You'll need to make just one change to make this section of program run on the Vic 20. Line 511Ø should read:

```
511Ø YS=ABS(YS-JA)
```

```
5ØØØ JA=LW/NX
5Ø1Ø XS=XA
5Ø2Ø FOR JB=Ø TO NX
5Ø25 YS=YA:XE=XS:YE=YA+LH
5Ø3Ø GOSUB 95ØØ
5Ø4Ø XS=XS+JA
5Ø5Ø NEXT JB
5Ø6Ø JA=LH/NY
5Ø7Ø YS=YA+LH
5Ø8Ø FOR JB=Ø TO NY
5Ø9Ø XS=XA+LW:XE=XA:YE=YS
51ØØ GOSUB 95ØØ
511Ø YS=YS-JA
512Ø NEXT JB
513Ø RETURN
```

```
5ØØØ DEF PROCGRID(XA,YA,LW,LH,NX,NY)
5Ø1Ø JA=LW/NX
5Ø2Ø XS=XA
5Ø3Ø FOR JB=Ø TO NX
5Ø7Ø PROCLINE(XS,YA,XS,YA+LH)
5Ø8Ø XS=XS+JA
5Ø9Ø NEXT JB
51ØØ JA=LH/NY
51Ø5 YS=YA+LH
511Ø FOR JB=Ø TO NY
515Ø PROCLINE(XA+LW,YS,XA,YS)
516Ø YS=YS-JA
517Ø NEXT JB
518Ø ENDPROC
```

Coordinates (XA,YB) specify the bottom left-hand corner of the grid. LW specifies the width, and LH the height. NX specifies the number of horizontal divisions, and NY the number of vertical divisions. The variable JA specifies the distance between the vertical lines, and the FOR ... NEXT loop draws horizontal lines stepped off in this distance. The second FOR...NEXT loop draws vertical lines in steps calculated by Line 5Ø6Ø (51ØØ for the Acorn).

The routine between Lines 5ØØØ and 518Ø draws horizontal lines in one step from left to right, and vertical lines in one step from bottom to top, so that such a grid can map only flat surfaces. It could not, for example, depict irregularities within the surface.

To display the grid on the screen, enter these lines to call the routine and RUN the program:

```
1ØØ GOSUB 9ØØØ
175 LET XA=Ø: LET YA=Ø: LET LW=255:
   LET LH=175: LET NX=16: LET NY=12
18Ø GOSUB 5ØØØ
19Ø STOP
```

```
1ØØ HIRES Ø,1:COLOUR 5,1
175 XA=Ø:YA=Ø:LW=32Ø:LH=2ØØ:
   NX=4:NY=4
18Ø GOSUB 5ØØØ
19Ø GOTO 19Ø
```

```
1ØØ GRAPHIC 2:COLOR 1,5,Ø,Ø
175 XA=Ø:YA=Ø:LW=1Ø23:LH=1Ø23:
   NX=4:NY=4
18Ø GOSUB 5ØØØ
19Ø GOTO 19Ø
```

```
1ØØ MODEØ
11Ø PROCINIT
18Ø PROCGRID(Ø,Ø,1279,1Ø23,2Ø,15)
19Ø END
```

(Line 11Ø is not necessary in this program at the moment, because Line 1ØØ achieves the same result. It is included for completeness, because it is expanded in a later program.)

```
1ØØ PMODE4:SCREEN1,1
1Ø5 PI=4*ATN(1)
11Ø GOSUB 9ØØØ
175 XA=Ø:YA=Ø:LW=255:LH=191:
   NX=4:NY=3
18Ø GOSUB 5ØØØ
19Ø GOTO 19Ø
```

When you RUN the program, you should see a grid filling the entire screen. To see how versatile the program is, make the changes below and RUN again:

```
175 LET XA=1Ø: LET YA=1Ø: LET
   LW=24Ø: LET LH=144: LET NX=1: LET
   NY=1
```

```
175 XA=1Ø:YA=1Ø:LW=3ØØ:LH=18Ø:
   NX=1:NY=1
```

```
175 XA=7:YA=7:LW=1ØØ7:LH=1ØØ7:
   NX=1:NY=1
```

```
18Ø PROCGRID(Ø,Ø,1279,1Ø23,1,1)
```

Q+A

Can I add colour to liven up my wireframe drawings?

Usually, wireframe drawings are in two colours only—most commonly black and white. One reason for this is that too many colours would complicate the image. More importantly, however, the addition of colour reduces the resolution of the image and in some computers, such as the Spectrum, can cause problems when one colour overprints another. On the Commodore 64, the use of colour would halve the drawing screen, so the program would have to be modified to compensate. Towards the end of this series of articles, we discuss the addition of colour further, but there is no reason you should not experiment with colour now.

```
175 XA=10:YA=10:LW=240:LH=160:
    NX=1:NY=1
```

This time, a rectangular box is drawn, because a grid with one horizontal and one vertical division is specified. By giving NX and NY suitable values, as above, you can draw a grid in which the number of horizontal divisions is different from the number of vertical divisions.

Make the changes below and RUN again:

```
175 LET XA=0: LET YA=0: LET LW=160:
    LET LH=144: LET NX=15: LET NY=10
```

```
175 XA=10:YA=80:LW=150:LH=90:
    NX=15:NY=10
```

```
175 XA=7:YA=500:LW=507:LH=507:
    NX=15:NY=10
```

```
180 PROCGRID(0,0,800,765,15,10)
```

```
175 XA=0:YA=31:LW=160:LH=160:
    NX=15:NY=10
```

The grid no longer fills the screen, but instead is a square in the bottom left-hand area. The square shape is achieved by giving LW and LH the appropriate value, and the number of divisions is specified by NX and NY as above.

DRAWING CIRCLES

A rectangular grid is not the only 'building block' you can use for wireframe drawing; it is often useful to be able to draw circles. Some micros have a command that lets you draw circles by specifying the centre and radius.

This direct command, however, does not give the degree of controllability you need for drawing three-dimensional images. With perspective, a circle in one view might be an ellipse in another, or some other curve in others. Although the CIRCLE command can be used to draw ellipses, it cannot usually cope with the third dimension, which is essential for natural-looking shapes. So it is better to be able to define a general function.

One way to draw a circle is with a series of short, straight-line sections. Provided the lines are short, the circumference of the circle will appear as a smooth curve, but the shorter the lines, the more of them you need and the longer will be the drawing time. Here is a routine to draw a circle radius R, with centre at (XS,YS); do not RUN the program yet:

```
6000 IF N=0 THEN LET N=20+INT (R/10)
6020 LET JA=2*PI/N
6050 LET XR=XS: LET YR=YS
6060 LET JB=0: LET XS=XS+R
6070 FOR J=2 TO N
6080 LET JB=JB+JA
6090 LET XE=XR+R*COS JB: LET
     YE=YR+R*SIN JB: GOSUB 9500
6100 LET XS=XE: LET YS=YE
6110 NEXT J
6120 LET XE=XR+R: LET YE=YR: GOSUB
     9500
6130 LET XS=XR: LET YS=YR
6160 RETURN
```

On the Commodores, change the PI in Line 6020 to the symbol π.

```
6000 IF N=0 THEN N=20+INT(R/10)
6020 JA=2*PI/N
6050 XR=XS: YR=YS
6060 JB=0:XS=XS+R
6070 FOR JC=2 TO N
6080 JB=JB+JA
6090 XE=XR+R*COS(JB):YE=YR+R*SIN
     (JB):GOSUB9500
6100 XS=XE:YS=YE
6110 NEXT JC
6120 XE=XR+R:YE=YR:GOSUB9500
6130 XS=XR:YS=YR
6160 RETURN
```

```
6000 DEF PROCCIRCLE(XS,YS,R,N)
6010 IF N=0 THEN N=20+INT(R/10)
6020 JA=2*PI/N
6050 PROCMOVE(XS+R,YS)
6060 JB=0
6070 FOR JC=2 TO N
6080 JB=JB+JA
6110 PROCDRAW(XS+R*COS(JB),
     YS+R*SIN(JB))
6120 NEXT JC
6150 PROCDRAW(XS+R,YS)
6160 ENDPROC
```

The variable N sets the number of straight-line segments to be used for the circumference of the circle. If you specify N=0, Line 6000 (6010 on the Acorn) calculates how many segments are needed for the smoothest circle, taking into account the size of the display.

Line 6020 calculates the angle of each line segment on the circumference. Line 6050 moves the cursor to a position on the circumference. The FOR... NEXT loop draws each line segment, except the last one, which is drawn by Line 6120 (6150 for Acorn) to ensure that the last line joins up with the first.

To see how the routine works, delete Line 180 and add the next few lines to call it:

```
150 FOR R=20 TO 70 STEP 10
155 LET XS=128: LET YS=102: LET N=24
160 GOSUB 6000
170 NEXT R
```

```
150 FOR R=20 TO 100 STEP 20
155 XS=160:YS=100: N=24
160 GOSUB 6000
170 NEXT R
```

```
150 FOR R=50 TO 500 STEP 100
155 XS=512:YS=512: N=24
160 GOSUB 6000
170 NEXT R
```

```
150 FOR R=60 TO 500 STEP 80
160 PROCCIRCLE(640,512,R,24)
170 NEXT R
```

```
150 FOR R=0 TO 100 STEP 20
155 XS=128:YS=102: N=24
160 GOSUB 6000
170 NEXT R
```

The display on your screen should now show a number of concentric circles at the centre of the screen. As with the Grid routine, you can vary the parameters in the program that calls the routine to change the display. Line 150 sets the radius of the first circle, and the amount by which it is increased to give the radii of successive circles. Line 155 (160 on the Acorn) specifies the centre of the circles and the number of line sections in the circumference. As an exercise, vary these values and note the effect on the display.

If you are wondering what has happened to the grid routine, it is still in the memory, but since you have rewritten the lines of code that call the routine, the computer does not display it. Routines like these can be collected into a library of useful line graphics routines for use as needed. They can be used in all sorts of graphics programs, as well as those for wireframe drawing. You can add new routines as you need them. Once they are in the computer, save them on tape or disk. When you want to use them, load them into memory again, merging several routines together if necessary (see pages 339 to 343).

Next time you'll see how to use these routines to create three-dimensional wireframe drawings.

COMMODORE TRACER

When you are all at sea with your programs and the error messages are flying, switch on this trace and get back on course, before you sink beneath a sea of syntax

It is almost impossible to key in a long program—like your assembler—without introducing some errors. No matter how much it is checked, there are some bugs that defy even the deftest programmer, without the aid of some powerful diagnostic tool.

Having a properly working assembler is essential. Many of the following chapters depend on it and it is vital that you locate all of the bugs in it now. So *INPUT* is providing Commodore 64 and Vic 20 owners with a trace program to help them check their assemblers out. The Dragon and Tandy have trace programs built in. So do the BBC and Electron. And a trace program for the Spectrum was given last time.

The trace program listed below is given in assembly language as well as machine code. If your assembler is not working you can feed in the trace program machine code using the machine code monitor given on pages 280 and 281. If your assembler is working, you can assemble the trace and SAVE it so that you can use it to diagnose problems in other BASIC programs that you have written. And if you are not sure whether your assembler is working or not, you can test it by trying to assemble the trace program.

HOW TO USE IT

When a BASIC program will not RUN, your computer will often give you an error message which tells you which line it cannot execute. This may be all you need to know to debug a short, simple program. But when your programs get longer and more complicated, such a message may still leave you in the dark. A particular line may be executed a number of times while the program is being RUN. And other lines that have been executed before it may set the variables to values that cause problems in the line which your computer eventually falters on. The articles on pages 334 to 338 and 375 to 379 showed how difficult it can be to de-bug a program.

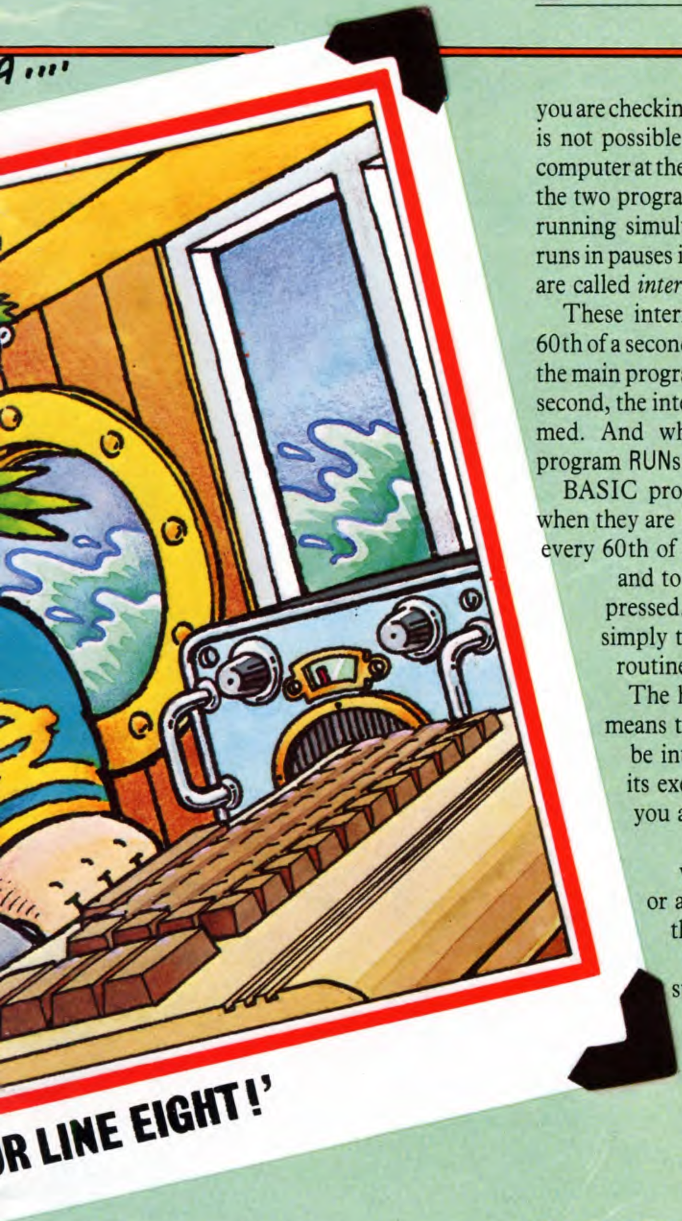
The trace program for the Commodore simply PRINTs out on the screen the number of each line as it is executed.

You need a copy of the program—either



printout or the version published in *INPUT*. Follow the program to the point where it stops, using the trace. This way you will be able to see clearly the structure of the program. You'll be able to spot whether the

- WHY YOU NEED A TRACE
- ENTERING THE TRACE TO CHECK YOUR ASSEMBLER
- WHAT A TRACE CAN DO
- TRACKING DOWN ERRORS
- HOW THE TRACE WORKS
- HOW TO USE IT



computer is RETURNing from subroutines properly. You'll also be able to work out the value of the variables as you go and check that conditional IF ... THEN statements are being fulfilled and that GOTOs go to the right line.

HOW IT WORKS

A trace program is rather special. It runs while another program—the BASIC program that you are checking out—is RUNning. Normally it is not possible to run two programs in your computer at the same time. In this case though, the two programs—although they seem to be running simultaneously—are not. The trace runs in pauses in the main program using what are called *interrupt driven routines*.

These interrupt the main program every 60th of a second on the Commodore 64. While the main program is halted for a fraction of the second, the interrupt driven routine is performed. And when it is finished, the main program RUNs again until the next interrupt.

BASIC programs are always interrupted when they are RUN. The computer breaks off every 60th of a second to scan the keyboard and to check to see if a key has been pressed. Interrupt driven routines are simply tacked onto this keyboard scan routine.

The high frequency of interruption means that a long line of BASIC may be interrupted several times during its execution. So the trace may give you a line number repeated several times. Conversely, if a line is very short—say a single PRINT or a RETURN from a subroutine—there is a slight chance that the trace will miss it. Sometimes, such lines take less than a 60th of a second and the interrupt could miss them. If the trace does not list the number of a short line, try adding a delay—a FOR ... NEXT loop or a REM statement—to it.

The following program prints out the number of the line of BASIC being executed as it is executed in the top left-hand corner of the screen. This is to keep it out of the way of anything that your BASIC program might want to PRINT on the screen.

If you key it in using your assembler, the origin is 49,152. If you are using your machine code monitor the start address is also 49,152. And to switch on the routine you use SYS49152.

When you RUN a program, the BASIC line numbers will be flipped through very rapidly. To slow it down to a readable speed, press the [F1] key. To stop it, hit the [RUN/STOP] key. This will leave the line number the machine stopped on displayed. Key in CONT, and your computer will start again from where it left off.

```
ORG 49152
SEI                 78
LDA #&0D            A9 0D
STA &0314           8D 14 03
LDA #&C0            A9 C0
STA &0315           8D 15 03
CLI                 58
RTS                 60
LDA #&01            A9 01
LDX &D021           AE 21 D0
CPX #&F1            E0 F1
BNE COL             D0 02
LDA #&00            A9 00
.□COL STA &FF       85 FF
LDA #&00            A9 00
STA &FE             85 FE
LDA &39             A5 39
STA &FB             85 FB
LDA &3A             A5 3A
STA &FC             85 FC
LDY #&07            A0 07
.□DIG LDX #&30      A2 30
.□FIG SEC           38
LDA &FB             A5 FB
SBC NUMS-1,Y        F9 75 C0
PHA                 48
DEY                 88
LDA &FC             A5 FC
SBC NUMS+1,Y        F9 77 C0
BCC OUT             90 09
STA &FC             85 FC
PLA                 68
STA &FB             85 FB
INX                 E8
INY                 C8
BNE FIG             D0 E8
.□OUT PLA           68
TXA                 8A
STY &FD             84 FD
INC &FE             E6 FE
LDY &FE             A4 FE
STA &0420,Y         99 20 04
LDA &FF             A5 FF
STA &D820,Y         99 20 D8
```

```
LDY &FD                     A4 FD
DEY                            88
BPL DIG                     1Ø D1
LDA &FB                     A5 FB
ORA #&3Ø                    Ø9 3Ø
STA &Ø425                8D 25 Ø4
LDA &FF                     A5 FF
STA &D825                8D 25 D8
LDA &C5                     A5 C5
CMP #&Ø4                    C9 Ø4
BNE QUIT                    DØ ØA
LDX #&ØØ                    A2 ØØ
.□AGAIN LDY #&ØØ            AØ ØØ
.□BACK INY                     C8
BNE BACK                    DØ FD
INX                            E8
BNE AGAIN                   DØ F8
.□QUIT JMP &EA31         4C 31 EA
.□NUMS WOR 1Ø               ØA ØØ
WOR 1ØØ                     64 ØØ
WOR 1ØØØ                    E8 Ø3
WOR 1ØØØØ                   1Ø 27
```

HOW IT WORKS

The first seven instructions form a small routine that switches the main program on. SEI SEts the Interrupt flag in the status register. This switches the normal interrupts off so that you can change the *interrupt vector* and point it to the trace routine. If you did not disable the interrupts while you were doing this, there would be a danger that an interrupt might occur half way through the change and be directed off to a place you did not intend.

LDA #&ØD and STA &Ø314, and LDA #&CØ and STA &Ø315 load the number CØØD into the interrupt vector, which occupies memory locations Ø314 and Ø1315 hex. The hash # sign tells the assembler what follows is a number rather than an address, and the ampersand & tells it that the number is in hex. There is no instruction to load a number directly into a memory location. First the number has to be loaded into either the A, X or Y register with an LDA, LDX or LDY instruction, then contents of the register are stored in the appropriate memory location with an STA, STX or STY instruction.

CØØD is the address of the beginning of the main trace routine which starts in the assembly language listings here with the instruction LDA #&Ø1.

CLI then CLears the Interrupt flag, in other words it enables the interrupts again. And RTS ReTurnS BASIC.

Once those seven instructions have been executed, the trace routine has been switched on. Now, every 60th of a second, when the computer interrupts its main operation to scan the keyboard, it is directed to the trace routine instead.

THE MAIN ROUTINE

You will notice that the main routine does not start by switching the interrupts off, as you might expect. After all you wouldn't want your interrupt routine to be interrupted, as that would put it into an endless loop. But on the 6510 chip (and the 6502) the interrupts are automatically disabled when an interrupt routine is started.

The first thing the main routine does do is check the background screen colour—there is no point in printing white numbers onto a white screen. You won't see them.

LDA #&Ø1 loads the A register with the number 1, which is the number that will give you white. LDX &DØ21 then loads the X register with the contents of memory location DØ21 hex, or 53,281 decimal. This location controls the background colour. The most significant four bits are always held high—that is set to 1. So if the background colour is white, the contents of that location will be 1111ØØØ1 in binary, 241 decimal or F1 hex. And CPX #&F1 ComPares the contents of the X register—that is the contents of DØ21—with F1.

If the background colour is white the CPX instruction sets the zero flag in the status register. And BNE—Branch if Not Equal—jumps over the instruction LDA #&ØØ to the next mention of the label COL if the zero flag is not set. But if the background colour is white and the zero flag is set, the jump is not executed. LDA #&ØØ then loads the ac-





cumulator with Ø, the number corresponding to black.

Whatever the result of this test, the contents of the accumulator are then stored in the memory location FF on the zero page. The microprocessor refers back to the number in that location when it gets round to putting the line number on the screen.

LDA # &ØØ and STA &FE sets the contents of the zero page memory location FE to Ø. This is going to be used as a counter when the routine works out the position of the decimal digits on the screen.

LDA &39 and STA &FB, and LDA &3A and STA &FC transfer the contents of memory locations 39 and 3A hex, 57 and 58 decimal, into FC and FB. Locations 57 and 58 hold the number of the BASIC line which the computer is currently executing. FC and FB are free locations on the zero page where these numbers can be manipulated—remember, these numbers are in hex and the routine has to do a considerable amount of work on them to print them out in decimal on the screen. You wouldn't want a trace program that gave you hex line numbers, would you?

The next section works out the decimal digits which will then be poked onto the screen.

LDY #&Ø7 loads the Y register with the number 7. This is going to be used as an offset and a counter, so the routine knows which decimal digit it is working on. And LDX #&3Ø, puts 48 decimal into the X register. 48 is the ASCII code for the figure Ø and, again, the X register is going to be used as a counter, counting along the ASCII codes for the correct digit to poke in that position on the screen.

SEC SEts the Carry flag. This should always be done before a subtraction is carried out because it is the 1 from the carry flag that gives the 1 that's added to the flipped bits to give 2's complement (see page 181). When the microprocessor subtracts, it flips the bits and adds. The addition takes the carry into account. So if the carry flag is set to 1, the 1 is added in effectively giving 2's complement. And when the carry flag is not set—that is, it's Ø—Ø is added in. That effectively subtracts an extra one.

This is very useful if you are subtracting the two-byte numbers. If you set the carry, subtract the low bytes first, then the high bytes. Then if the low bytes need a borrow the carry flag will be set to Ø. When the high bytes are subtracted, an extra 1 will be subtracted and the borrow will automatically be accounted for. You will note that the carry flag works the other way round from what you would expect logically. When it is set to 1 there is no borrow, and when it is reset to Ø, there is.

So the rule is: If you're going to subtract, set the carry flag. And if you're going to add, reset the carry flag to Ø—you don't want an extra 1 added in then, after all.

The next thing that happens in this routine is exactly that sort of two-byte subtraction routine. LDA &FB loads the accumulator with

the contents of FB, which are the low byte of the current line number. SBC NUMS−1,Y subtracts the low byte of a number that it looks up in the data table which starts at the label NUMS. The table comprises the WOR numbers at the end of the routine. Each of the decimal numbers given there are loaded, as hex, into two bytes. The SBC instruction looks up the byte you want to subtract, by jumping to the byte before the label NUMS—in other words, memory location NUMS−1—then counting the number contained in the Y register on from there. So on the first pass, when the contents of Y are 7, it subtracts the byte six (7 − 1) on from NUMS, which is the low byte of WOR 1ØØØØ. It may seem a little strange to use two offsets like this—the −1 with the label and the contents of the Y register—but this saves instructions and gives the correct value of the Y register at the end of this routine. It's being used to memorize which decimal digit the routine's working on.

The result of the first subtraction is pushed onto the stack with PHA. The contents of the Y register are then decremented by DEY. And LDA &FC and SBC NUMS+1,Y subtract the high byte of the same decimal in the table.

BCC means Branch on Carry Clear. So if there is carry—or in this case, a borrow—from the subtraction, the routine branches to the next mention of the label OUT, which appears before PLA. Remember that the carry flag is clear, or Ø, when there is a borrow. And if there is no borrow and the carry flag is set to 1, the routine proceeds to the next instruction.

STA &FC puts the remainder of this high byte back in FC. And PLA and STA &FB pulls the remainder of the low byte off the stack and puts it back in FB. INX increments the X register. The effect of this is to make the X register count along the ASCII codes of the decimal digits.

INY increments the Y register. This is done because the register was decremented before and the value in the Y register needs to be restored before the next instruction BNE FIG sends the processor round this loop again.

SAVING BYTES

BNE means Branch if result Not Equal to zero. And the result of incrementing the Y register could only be equal to zero if it had been −1 before. If you examine how this program works in detail you will see that this never happens. But using a branch, which uses relative addressing, rather than an unconditional jump, which uses absolute addressing, saves one byte.

This branch means that the microprocessor goes round and round this loop, taking away the decimal values it got from the data table incrementing the X register—and counting along the ASCII codes—as it goes. As you can see, the way this routine works out is by counting how many times it can take away 10,000, how many times it can take away 1,000, how many times it can take away 100 and how many times it can take away ten. When it does these repeated subtractions and finds it can't go, the BCC instruction branches out of the loop to the instruction PLA. This pulls the low byte off the stack into the A register. It is immediately disposed of by transferring the contents of the X register into the accumulator with the instruction TXA.

The only reason that the low byte is pulled off the stack is to prevent the stack growing.

As the subtraction will no longer go, the X register now contains the ASCII value for the correct digit to put into the decimal place that is being worked out. The TXA puts that value into the accumulator, ready for outputting.

PUTTING IT ON THE SCREEN

STY &FD puts the decimal place counter into temporary storage in FD, so that the Y register can be used for something else. The position counter in FE is then incremented and its new value is loaded into the Y register.

STA &Ø42Ø,Y is the instruction that actually puts the digit onto the screen. Memory location Ø42Ø is in the screen area—the top right-hand part of it. The ASCII code of the digit the routine has worked out is stored in the succeeding memory locations by adding the offset Y. So the right digit is poked onto the screen in the right place.

LDA &FF then loads the accumulator with the contents of FF which are Ø if black letters are required and 1 if white are required. The colour was worked out before.

That value is then stored in the appropriate location in the colour file. The screen memory and the colour memory work entirely independently. What is displayed in any screen location is completely independent of what colour it is. And the fact that the colour is fixed after the digit has been poked onto the screen does not matter. As the whole routine takes less than a 60th of a second, there is no chance that you will see a digit appearing on the screen and then, a fraction of a second later, changing colour. Your eyes cannot respond that fast. Neither can the TV screen.

The Y register is loaded with the contents of FD again which is then decremented. This is the decimal position store which counts from 7 downwards.

BPL means Branch on PLus result—and zero counts as a plus. So BPL DIG sends the microprocessor back to the beginning of this whole routine if it is not on the second to last digit. In other words, it loops back to work out the next digit unless the last digit was the tens. The units are worked out by a separate routine which follows immediately.

LDA &C5 loads the accumulator with whatever's left after the tens have been worked out. It is then ored with 48 to give the ASCII. An ORA instruction is used rather than an ADC—ADd with Carry—as you would have to clear the carry flag before adding. So ORing, which is not affected by the carry flag, saves a byte.



The result is then stored on the screen in the units position and the correct colour is added.

The zero page memory location C5 contains details of the last key that's been pressed— the actual number stored here depends on the key's position on the keyboard rather than the corresponding ASCII.

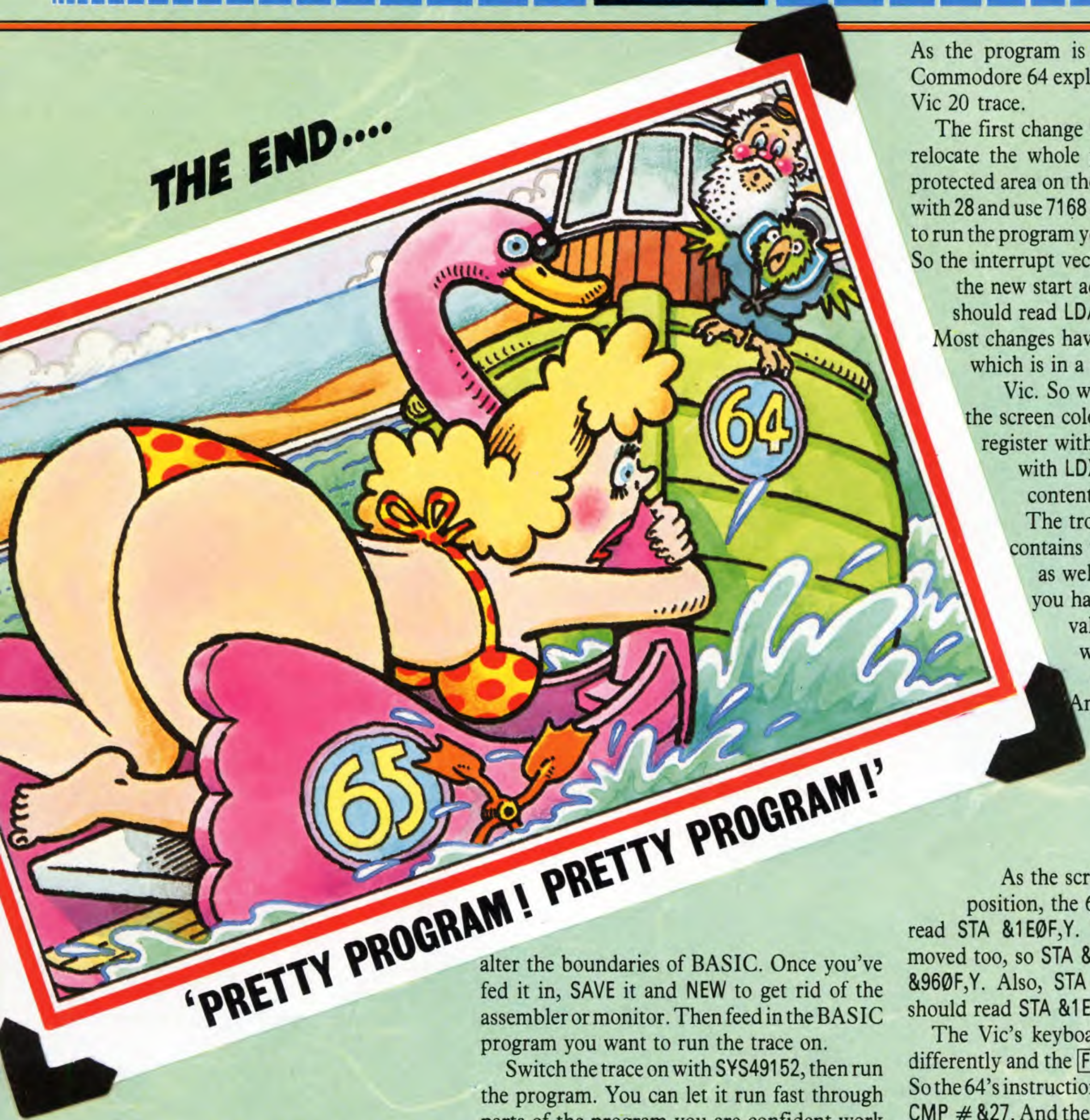
The contents of C5 are loaded into the accumulator and CoMPared with 4—4 corresponds to the [F1] key. BNE QUIT then jumps over the delay routine that follows if the [F1] key has not been pressed, and the result of CMP &Ø4 is not Ø.

The delay routine consists of a loop within a loop. The X register and the Y register are both loaded with Ø and incremented. And the microprocessor is branched back each time the result of the incrementation is not zero. Obviously, you'll only get a zero result when the register had filled up and overflowed. So it goes round each loop 256 times—this means that the inner Y-register loop is executed 256 × 256 times.

Memory location EA31 is the beginning of the regular interrupt routine. JMP &EA31 sends the microprocessor off to perform the regular interrupt routine. If you did not do this, the keyboard would freeze up and you would not be able to use your Commodore.

What follows is the decimal data used to work out the digits. This data is entered as WORds. WOR is not assembly language. It is an assembler instruction which tells the assembler to leave two bytes empty for data.

You will have noticed that there is no return instruction in this program, so you may be wondering how it returns to BASIC. On the other hand, you may have guessed that the return instruction is in the regular interrupt



routine. Once it has performed its normal duties—like scanning the keyboard—it automatically switches the interrupts back on and returns directly to BASIC without returning to the routine you've keyed in. Then the computer does another 60th of a second's worth of BASIC and performs this interrupt routine all over again.

HOW TO USE IT

To check out your assembler—or any other program you want to run a trace on—you must first have the trace routine in the computer's memory. If your assembler is working you can assemble the routine in the normal way. Otherwise, key in the hex numbers given using your machine code monitor (see page 280).

You will notice that the trace routine is in the protected area after CØØØ, so you don't have to alter the boundaries of BASIC. Once you've fed it in, SAVE it and NEW to get rid of the assembler or monitor. Then feed in the BASIC program you want to run the trace on.

Switch the trace on with SYS49152, then run the program. You can let it run fast through parts of the program you are confident work properly. Press the [F1] key when you approach parts you're having difficulty with. And hit the [RUN/STOP] key if you want to freeze the trace.

Make sure that you have a listing of the program beside you when you are doing a trace, so that you can see what is happening—otherwise the numbers flashing up on the top right-hand corner of the screen will mean nothing to you.

If the program calls for you to input something on the top line, use the second line instead as you won't be able to write over the trace's digits.

To switch the trace off press [RUN/STOP] and [RESTORE].

The Commodore 64 trace program will run on the Vic 20 too, with a few minor modifications. As the program is basically the same, the Commodore 64 explanation also holds for the Vic 20 trace.

The first change that has to be made is to relocate the whole program. CØØØ is not a protected area on the Vic. So POKE 52 and 56 with 28 and use 7168 as your start address. And to run the program you'll have to call SYS7168. So the interrupt vector has been changed to the new start address. The second LDA should read LDA # &1C.

Most changes have to do with the screen, which is in a different position in the Vic. So when you want to look at the screen colour you must load the X register with the contents of 9ØØF—with LDX &9ØØF—instead of the contents of DØ21—LDX &DØ21. The trouble is 9ØØF on the Vic contains the colour of the border as well as the screen itself. So you have to check whether the value there is less than 16, which means that the ink colour should be white. Anyway, the 64's CPX &F1 and BNE COL must be replaced with:

```
CPX # &1Ø
BCC COL
```

As the screen area is in a different position, the 64's STA &Ø4ØF,Y should read STA &1EØF,Y. The colour memory is moved too, so STA &D82Ø,Y should read STA &96ØF,Y. Also, STA &Ø425 and STA &D825 should read STA &1E14 and STA &9614.

The Vic's keyboard is organized slightly differently and the [F1] does not give 4 but 39. So the 64's instruction CMP # &Ø4 should read CMP # &27. And the regular interrupt routine begins at EABF on the Vic, rather than EA31. So JMP &EA31 should read JMP &EABF.

Otherwise the Vic's trace operates in exactly the same way as the 64's.

Of course, you will have to hand assemble the Vic's trace, unless you have bought a commercial assembler. But the following is the machine code version of the program, so you can check your conversion or simply key the hex version straight in.

```
78 A9 ØD 8D 14 Ø3 A9 1C 8D 15 Ø3 58 6Ø A9
Ø1 AE ØF 9Ø EØ 1Ø 9Ø Ø2 A9 ØØ 85 FF A9 ØØ
85 FE A5 39 85 FB A5 3A 85 FC AØ Ø7 A2 3Ø
38 A5 FB F9 75 1C 48 88 A5 FC F9 77 1C 9Ø
Ø9 85 FC 68 85 FB E8 C8 DØ E8 68 8A 84 FD
E6 FE A4 FE 99 ØF 1E A5 FF 99 ØF 96 A4 FD
88 1Ø D1 A5 FB Ø9 3Ø 8D 14 1E A5 FF 8D 14
96 A5 C5 C9 27 DØ ØA A2 ØØ AØ ØØ C8 DØ FD
E8 DØ F8 4C BF EA ØA ØØ 64 ØØ E8 Ø3 1Ø 27
```

COMPUTER CONVERSION TABLES

Do you have trouble working out how many inches there are in a metre? Or how many pints in a litre? If so, then the *INPUT* conversion program has all the answers

Working out how many litres of petrol your car can hold, or how to buy metric tiles that are priced by the square yard can be a difficult task unless you happen to know the various metric conversion factors by heart. The programs below are an easy way to convert metric units into Imperial, and Imperial units into metric, and work for any of the standard units you are likely to come across.

WHICH MEASUREMENT?

All the common measurements are included in the program, along with some you'll need less often—such as millimetres of mercury, a measure of pressure (abbreviated to mmHG in the program).

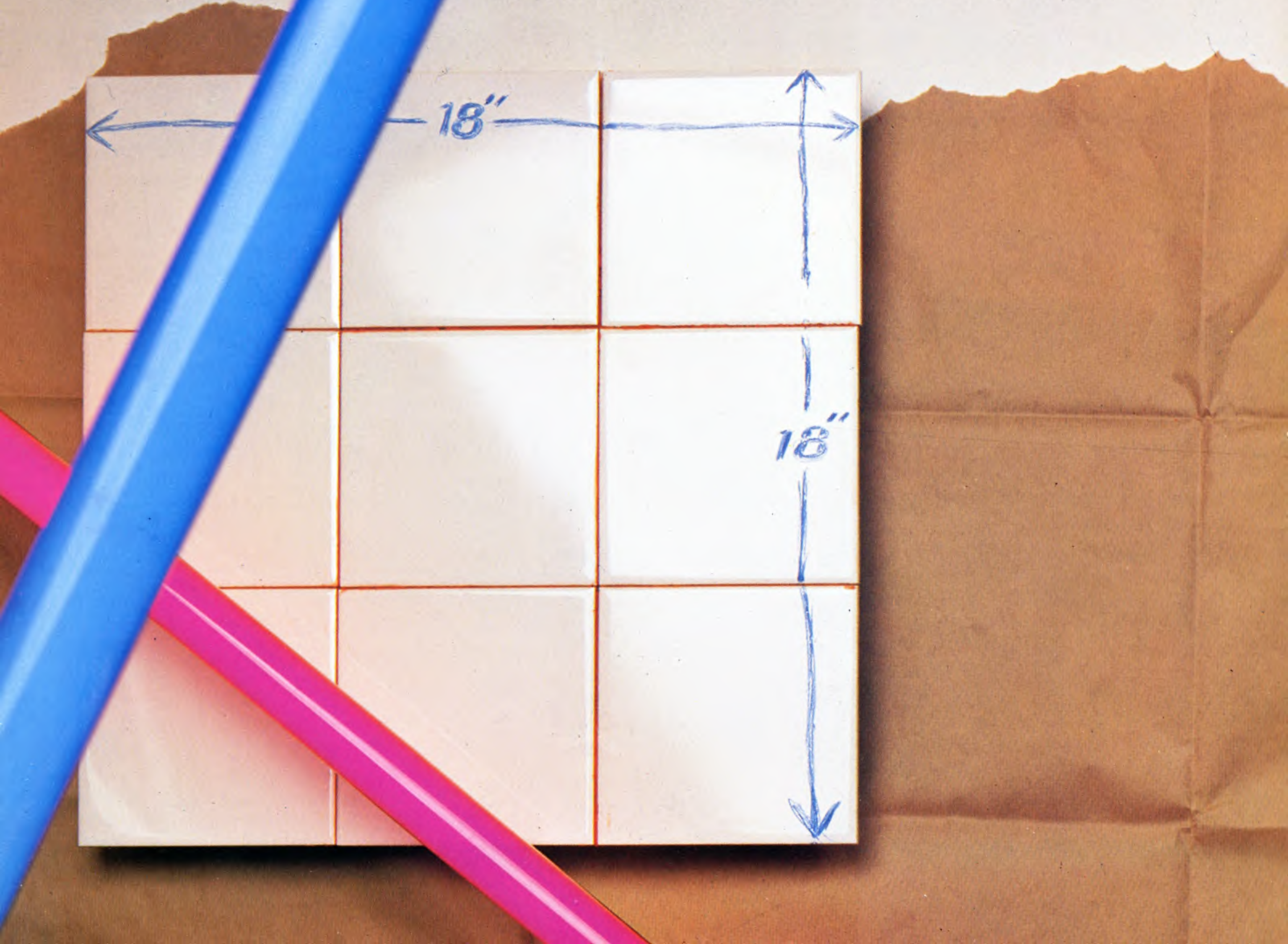
First, type in the program and RUN it. You are shown a menu of options. The first option (Quit) allows you to return to BASIC, while all the others refer to the type of units you want to convert. The choices available are: length, area, volume, weight, pressure, and temperature. Strictly speaking, the 'weight' option gives measurements of mass, but conforms to popular usage.

To the left of each option is a number. To select a popular option just press the key with the relevant number. For example, to convert a unit of length (option 1), press the '1' key. It does not matter at this point whether you want to change a metric unit into an Imperial one, or the other way round: that comes later.

WHICH UNITS?

Once you have chosen the type of measurement you want to change, and pressed the relevant key, the computer PRINTs a second menu. This allows you to choose the units you want to convert.

So, if you pressed the '1' key to convert units of length you now get a menu which lists all the possible units (inches, feet, millimetres, centi-



- CONVERTING METRIC TO IMPERIAL MEASUREMENTS AND VICE VERSA
- HOW TO USE THE PROGRAM
- CONVERSIONS FOR LENGTH, AREA, WEIGHT, VOLUME, PRESSURE AND TEMPERATURE
- CONVERTING MIXED UNITS AND FRACTIONS

metres, and so on). If you want to convert inches into metric units, press the 1 key again—the number beside the 'inches' on the screen. Now enter the number of inches and hit ENTER or RETURN.

PRINTING THE ANSWERS

The computer works out the conversions and PRINTs the result in all the units of the other set (metric or Imperial). So if you typed in a value in inches, you would be given the equivalent in millimetres, centimetres, metres, and kilometres.

After you have been shown the converted values, the computer waits for you to press a key before continuing. If you press the ENTER or RETURN key, you are returned to the main menu to choose another option. If you press any other key, you are sent to the second menu—the computer assumes you want the same type of measurement again.

You can stop the program if you want by returning to the main menu, and pressing 'Ø' for the QUIT option.

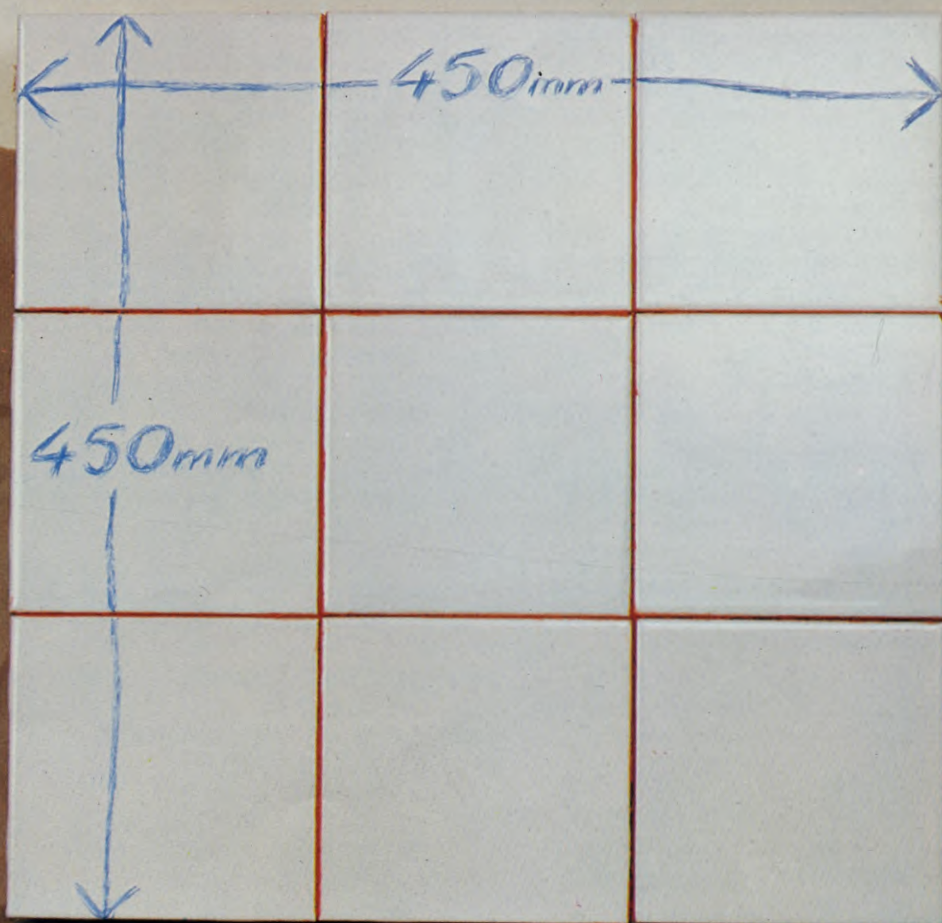
MIXED UNITS

Sometimes you may want to convert a value of an Imperial measure which is a mixture of units. For example, you might want to convert 2 feet 6 inches into metric units. You can do one of two things when this occurs.

The program accepts numbers which are not whole numbers: so, with the example above, if you know how many inches there are in a foot, you could work out a decimal fraction and tell the computer to convert that. With six inches, of course, there is nothing to worry about, six inches is exactly half a foot. So you would INPUT 2.5 feet.

If the fractions are difficult to work out you should use the second method. This is to convert the number in two stages. First, convert the feet into metric, and then convert the inches. All you do after that is add up the metric equivalent of both conversions (remember to add up the values of the same metric units).

If you have a metric fraction to convert, you



Do Continental tiles cover the same areas as British ones? If you're not sure, the INPUT conversion program will work out the calculations for you

can do the same again. But since metric units are conveniently scaled so that each consecutive unit is ten times larger, (or smaller) than the last, the problem is much easier to solve.

```
5 POKE 23658,8
1Ø DIM L(8): DIM L$(8,12): DIM A(7): DIM
   A$(7,9): DIM V(7): DIM V$(7,12): DIM
   M(6): DIM M$(6,9): DIM P(5): DIM
   P$(5,11)
2Ø FOR K=1 TO 8: READ L(K),L$(K): NEXT K
3Ø DATA 1,"INCHES",12,"FEET",36,
   "YARDS",6336Ø,"MILES",
   .Ø3937,"MILLIMETRES",
   .3937,"CENTIMETRES",39.37,
   "METRES",3937Ø,"KILOMETRES"
4Ø FOR K=1 TO 7: READ A(K),A$(K): NEXT K
5Ø DATA 1,"SQ INCHES",144,"SQ
   FEET",627264Ø,"ACRES",4.Ø145E9,
   "SQ MILES",.155,"SQ CMS",155Ø,
   "SQ METRES",1.55E7,"HECTARES"
6Ø FOR K=1 TO 7: READ V(K),V$(K): NEXT K
7Ø DATA 1,"CUBIC INCHES",1728,
   "CUBIC FEET",34.67, "PINTS",
   277.36,"GALLONS",.Ø61Ø2,"CC'S",
   61.Ø24,"LITRES",61Ø24,"CUBIC
   METRES"
8Ø FOR K=1 TO 6: READ M(K),M$(K):
   NEXT K
9Ø DATA 1,"OUNCES",16,"POUNDS",
   3584Ø,"TONS",.Ø3527,"GRAMS",
   35.27,"KILOGRAMS",3527Ø,"TONNES"
1ØØ FOR K=1 TO 5: READ P(K),P$(K):
   NEXT K
11Ø DATA 1,"PSI",51.73,"mmHG",6895,
   "N/SQ METRE",.Ø681,
   "ATMOSPHERES",68.95,"MILLIBARS"
12Ø CLS : PRINT INVERSE 1;''TAB 6;"WHICH
   CATEGORY (Ø-6) ?";TAB 31;"□"
13Ø PRINT AT 6,8;"Ø:- QUIT
   PROGRAM";AT 8,8;"1:- LENGTH";
   AT 1Ø,8;"2:- AREA";AT 12,8;
   "3:- VOLUME";AT 14,8;"4:- WEIGHT";
   AT 16,8;"5:- PRESSURE";AT 18,8;
   "6:- TEMPERATURE"
14Ø LET Z$=INKEY$: IF Z$<"Ø" OR
   Z$>"6" THEN GOTO 14Ø
15Ø IF Z$="Ø" THEN CLS : STOP
16Ø CLS : GOSUB 1ØØØ+(VAL Z$-1)*5ØØ
17Ø GOTO 12Ø
1ØØØ PRINT INVERSE 1;AT
   Ø,12;"□LENGTH□": PRINT AT
   2,6;"SELECT ORIGINAL UNITS": PRINT :
   FOR K=1 TO 8: PRINT 'TAB
   1Ø;K;":- ";L$(K): NEXT K
1Ø1Ø LET B$=INKEY$: IF B$<"1" OR
   B$>"8" THEN GOTO 1Ø1Ø
1Ø2Ø LET B=VAL B$: INPUT "INPUT
   NUMBER OF□";(L$(B)),VL
1Ø4Ø CLS : PRINT AT 2,4;VL;"□";
   L$(B);"□EQUALS"
1Ø5Ø IF B>4 THEN GOTO 1Ø8Ø
1Ø6Ø FOR K=1 TO 4: PRINT AT K*2+4,3;
   VL*L(B)/L(K+4);TAB 18;L$(K+4):
   NEXT K
1Ø7Ø GOTO 1Ø9Ø
1Ø8Ø FOR K=1 TO 4: PRINT AT K*2+4,3;
   VL*L(B)/L(K);TAB 18;L$(K): NEXT K
1Ø9Ø LET Z$=INKEY$: IF Z$="" THEN
   GOTO 1Ø9Ø
11ØØ IF Z$=CHR$ 13 THEN RETURN
111Ø CLS : GOTO 1ØØØ
15ØØ PRINT INVERSE 1;AT Ø,13;
   "□AREA□": PRINT AT 2,6;"SELECT
   ORIGINAL UNITS": PRINT : FOR K=1 TO
   7: PRINT 'TAB 1Ø;K;":- ";A$(K): NEXT K
151Ø LET B$=INKEY$: IF B$<"1" OR
   B$>"7" THEN GOTO 151Ø
152Ø LET B=VAL B$: INPUT "INPUT
   NUMBER OF□";(A$(B)),VL
154Ø CLS : PRINT AT 2,4;VL;"□";
   A$(B);"□EQUALS"
155Ø IF B>4 THEN GOTO 158Ø
156Ø FOR K=1 TO 3: PRINT AT K*2+4,3;
   VL*A(B)/A(K+4);TAB 18;A$(K+4):
   NEXT K
157Ø GOTO 159Ø
158Ø FOR K=1 TO 4: PRINT AT K*2+4,3;
   VL*A(B)/A(K);TAB 18;A$(K): NEXT K
159Ø LET Z$=INKEY$: IF Z$="" THEN
   GOTO 159Ø
16ØØ IF Z$=CHR$ 13 THEN RETURN
161Ø CLS : GOTO 15ØØ
2ØØØ PRINT INVERSE 1;AT Ø,12;
   "□VOLUME□": PRINT AT 2,6;
   "SELECT ORIGINAL UNITS": PRINT : FOR
   K=1 TO 7: PRINT 'TAB
   1Ø;K;":- ";V$(K): NEXT K
2Ø1Ø LET B$=INKEY$: IF B$<"1" OR
   B$>"7" THEN GOTO 2Ø1Ø
2Ø2Ø LET B=VAL B$: INPUT "INPUT
   NUMBER OF□";(V$(B)),VL
2Ø4Ø CLS : PRINT AT 2,4;VL;"□";
   V$(B);"□EQUALS"
2Ø5Ø IF B>4 THEN GOTO 2Ø8Ø
2Ø6Ø FOR K=1 TO 3: PRINT AT K*2+4,3;
   VL*V(B)/V(K+4);TAB 18;
   V$(K+4): NEXT K
2Ø7Ø GOTO 2Ø9Ø
2Ø8Ø FOR K=1 TO 4: PRINT AT K*2+4,3;
   VL*V(B)/V(K);TAB 18;V$(K):
   NEXT K
2Ø9Ø LET Z$=INKEY$: IF Z$="" THEN
   GOTO 2Ø9Ø
21ØØ IF Z$=CHR$ 13 THEN RETURN
211Ø CLS : GOTO 2ØØØ
25ØØ PRINT INVERSE 1;AT Ø,13;
   "□WEIGHT□": PRINT AT 2,6;
   "SELECT ORIGINAL UNITS": PRINT : FOR
   K=1 TO 6: PRINT 'TAB
   1Ø;K;":- ";M$(K): NEXT K
251Ø LET B$=INKEY$: IF B$<"1" OR
   B$>"6" THEN GOTO 251Ø
252Ø LET B=VAL B$: INPUT "INPUT
   NUMBER OF□";(M$(B)),VL
254Ø CLS : PRINT AT 2,4;VL;"□";
   M$(B);"□EQUALS"
255Ø IF B>3 THEN GOTO 258Ø
256Ø FOR K=1 TO 3: PRINT AT K*2+4,3;
   VL*M(B)/M(K+3);TAB 18;
   M$(K+3): NEXT K
257Ø GOTO 259Ø
258Ø FOR K=1 TO 3: PRINT AT K*2+4,3;
   VL*M(B)/M(K);TAB 18;M$(K):
   NEXT K
259Ø LET Z$=INKEY$: IF Z$="" THEN
   GOTO 259Ø
26ØØ IF Z$=CHR$ 13 THEN RETURN
261Ø CLS : GOTO 25ØØ
3ØØØ PRINT INVERSE 1;AT Ø,11;
   "□PRESSURE□": PRINT AT 2,6;
   "SELECT ORIGINAL UNITS": PRINT : FOR
   K=1 TO 5: PRINT 'TAB
   1Ø;K;":- ";P$(K): NEXT K
3Ø1Ø LET B$=INKEY$: IF B$<"1" OR
   B$>"5" THEN GOTO 3Ø1Ø
3Ø2Ø LET B=VAL B$: INPUT "INPUT
   NUMBER OF□";(P$(B)),VL
3Ø4Ø CLS : PRINT AT 2,4;VL;"□";
   P$(B);"□EQUALS"
3Ø5Ø LET T=Ø: FOR K=1 TO 5: IF K=B
   THEN GOTO 3Ø7Ø
3Ø6Ø PRINT AT K*2+4,3;VL*P(K)/
   P(B);TAB 18;P$(K): LET T=T+1
3Ø7Ø NEXT K
3Ø8Ø LET Z$=INKEY$: IF Z$="" THEN
   GOTO 3Ø8Ø
3Ø9Ø IF Z$=CHR$ 13 THEN RETURN
31ØØ CLS : GOTO 3ØØØ
35ØØ PRINT INVERSE 1;AT Ø,9;
   "□TEMPERATURE□": PRINT AT 3,11;
   "CONVERT:-": PRINT '"□□□
   CENTIGRADE TO FAHRENHEIT
   (C)□□OR FAHRENHEIT TO
   CENTIGRADE (F)"
351Ø LET B$=INKEY$: IF B$<>"C" AND
   B$<>"F" THEN GOTO 351Ø
352Ø IF B$="C" THEN GOTO 356Ø
353Ø INPUT "INPUT DEGREES
   FAHRENHEIT",VL
354Ø CLS : PRINT AT 1,2;VL;
   "□DEGREES FAHRENHEIT EQUALS"
355Ø PRINT 'TAB 2;(VL-32)*5/9;
   "□DEGREES CENTIGRADE": GOTO 359Ø
356Ø INPUT "INPUT DEGREES
   CENTIGRADE",VL
357Ø CLS : PRINT AT 1,2;VL;
   "□DEGREES CENTIGRADE EQUALS"
358Ø PRINT 'TAB 2;32+VL*9/5;
   "□DEGREES FAHRENHEIT"
359Ø PAUSE Ø: LET Z$=INKEY$: IF Z$=""
   THEN GOTO 359Ø
```




```
3600 IF Z$=CHR$ 13 THEN RETURN
3610 CLS : GOTO 3500
```

The Commodore program works on both the Commodore 64 and the Vic 20—but the program needs more than 3½K of memory to RUN so the Vic needs a memory expansion pack.

Since the program works on both Commodore computers, the Commodore 64 is formatted to use only the left-hand part of the display: only the first 22 characters in each line on the screen are used, as the Vic only has 22 characters in its lines.

```
10 DIM L(7),L$(7),A(6),A$(6),V(6),
   V$(6),M(5),M$(5),P(4),P$(4)
20 FOR K=Ø TO 7:READ L(K),L$(K): NEXT K
30 DATA 1,INCHES,12,FEET,36,YARDS,
   6336Ø,MILES,.Ø3937,
   MILLIMETRES,.3937
35 DATA CENTIMETRES,39.37,METRES,
   3937Ø,KILOMETRES
40 FOR K=Ø TO 6:READ A(K),A$(K): NEXT K
50 DATA 1,SQ INCHES,144,SQ FEET,
   627264Ø,ACRES,4.Ø145E9,SQ
   MILES,.155
55 DATA SQ CMS,155Ø,SQ METRES,
   1.55E7,HECTARES
60 FOR K=Ø TO 6:READ V(K),V$(K):
   NEXT K
70 DATA 1,CUBIC INCHES,1728,CUBIC
   FEET,34.67,PINTS,277.36,
   GALLONS,.Ø61Ø2
75 DATA CC'S,61.Ø24,LITRES,61Ø24,
   CUBIC METRES
80 FOR K=Ø TO 5:READ M(K),M$(K):
   NEXT K
90 DATA 1,OUNCES,16,POUNDS,3584Ø,
   TONS,.Ø3527,GRAMS,35.27,
   KILOGRAMS
95 DATA 3527Ø,TONNES
100 FOR K=Ø TO 4:READ P(K),P$(K):
   NEXT K
110 DATA 1,PSI,51.73,MMHG,6895,N/SQ
   METRE,.Ø681,ATMOSPHERES
115 DATA 68.95,MILLIBARS
120 PRINT "♥WHICH CATEGORY (Ø-5)?"
130 PRINT"■Ø- QUIT PROGRAM":
   PRINT"■1- LENGTH":PRINT
   "■2- AREA":PRINT"■3- VOLUME"
135 PRINT"■4- WEIGHT":PRINT
   "■5- PRESSURE":PRINT
   "■6- TEMPERATURE"
140 GET A$:IF A$<"Ø" OR A$>"6" THEN
   140
150 IF A$="Ø" THEN PRINT "♥":END
160 PRINT "♥":ON VAL(A$) GOSUB 1ØØØ,
   15ØØ,2ØØØ,25ØØ,3ØØØ,35ØØ
170 GOTO 12Ø
1000 PRINT TAB(7);"■LENGTH":
   PRINT "■SELECT ORIGINAL UNITS":FOR
   K=Ø TO 7
1005 PRINT "■";K+1;"-□";L$(K):
   NEXT K
1010 GET B$:IF B$<"1" OR B$>"8"
   THEN 1Ø1Ø
1020 B=VAL(B$)-1:PRINT "♥INPUT
   NUMBER OF□":PRINT L$(B);
1030 INPUT VL
1040 PRINT "♥";VL:PRINT L$(B);
   "□EQUALS■"
1050 IF B>3 THEN 1Ø8Ø
1060 FOR K=Ø TO 3:PRINT "■";VL*L
   (B)/L(K+4);TAB(23);"■";
   L$(K+4):NEXT K
1070 GOTO 1Ø9Ø
```

```
1080 FOR K=Ø TO 3:PRINT "■";VL*L
   (B)/L(K);TAB(23);"■";
   L$(K):NEXT K
1090 GET A$:IF A$="" THEN 1090
1100 IF A$=CHR$(13) THEN RETURN
1110 PRINT "♥":GOTO1000
1500 PRINT TAB(7);"■AREA":PRINT
   "■SELECT ORIGINAL UNITS":
   FOR K=Ø TO 6
1505 PRINT "■";K+1;"-□";A$(K):NEXT
   K
1510 GET B$:IF B$<"1" OR B$>"7" THEN
   1510
1520 B=VAL(B$)-1:PRINT "♥INPUT THE
   NUMBER OF□":PRINT A$(B);
1530 INPUT VL
1540 PRINT "♥";VL:PRINT A$(B);
   "□EQUALS■"
1550 IF B>3 THEN 1580
1560 FOR K=Ø TO 2:PRINT "■";VL*A
   (B)/A(K+4);TAB(23);"■";
   A$(K+4):NEXT K
1570 GOTO 1590
1580 FOR K=Ø TO 3:PRINT "■";VL*A
   (B)/A(K);TAB(23);"■";
   A$(K):NEXT K
1590 GET A$:IF A$="" THEN 1590
1600 IF A$=CHR$(13) THEN RETURN
1605 PRINT "♥":GOTO 1500
2000 PRINT TAB(7);"■VOLUME":PRINT
   "■SELECT ORIGINAL UNITS": FOR
   K=Ø TO 6
2005 PRINT "■";K+1;"-□";V$(K):
   NEXT K
2010 GET B$:IF B$<"1" OR B$>"7"
   THEN 2010
2020 B=VAL(B$)-1:PRINT "♥INPUT THE
   NUMBER OF □":PRINT V$(B);
2030 INPUT VL
2040 PRINT "♥";VL:PRINT V$(B);
   "□EQUALS■"
2050 IF B>3 THEN 2080
2060 FOR K=Ø TO 2:PRINT "■";VL*V
   (B)/V(K+4);TAB(23);"■";
   V$(K+4):NEXT K
2070 GOTO 2090
2080 FOR K=Ø TO 3:PRINT "■";VL*V
   (B)/V(K);TAB(23);"■";V$(K):
   NEXT K
2090 GET A$:IF A$="" THEN 2090
2100 IF A$=CHR$(13) THEN RETURN
2105 PRINT "♥":GOTO 2000
2500 PRINT TAB(7);"■WEIGHT":PRINT
   "■SELECT ORIGINAL UNITS":
   FOR K=Ø TO 5
2505 PRINT "■";K+1;"-□";M$(K):
   NEXT K
2510 GET B$:IF B$<"1" OR B$>"6" THEN
   2510
2520 B=VAL(B$)-1:PRINT "♥INPUT THE
   NUMBER OF□":PRINT M$(B);
```




```
2530 INPUT VL
2540 PRINT "♥";VL:PRINT M$(B);
   "□EQUALS■"
2550 IF B>2 THEN 2580
2560 FOR K=Ø TO 2:PRINT "■";VL*M
   (B)/M(K+3);TAB(23);"■";
   M$(K+3):NEXT K
2570 GOTO 2590
2580 FOR K=Ø TO 3:PRINT "■";VL*M
   (B)/M(K);TAB(23);"■";M$(K):
   NEXT K
2590 GET A$:IF A$="" THEN 2590
2600 IF A$=CHR$(13) THEN RETURN
2605 PRINT "♥":GOTO 2500
3000 PRINT TAB(7);"■PRESSURE":
   PRINT "■SELECT ORIGINAL UNITS":FOR
   K=Ø TO 4
3005 PRINT "■";K+1;"-□";P$(K):
   NEXT K
3010 GET B$:IF B$<"1" OR B$>"5"
   THEN 3010
3020 B=VAL(B$)-1:PRINT "♥INPUT THE
   NUMBER OF□":PRINT P$(B);
3030 INPUT VL
3040 PRINT "♥";VL:PRINT P$(B);
   "□ EQUALS■"
3050 T=Ø:FOR K=Ø TO 4:IF K=B THEN
   3070
3060 PRINT "■";VL*P(K)/P(B);
   TAB(23);"■";P$(K):T=T+1
3070 NEXT K
3080 GET A$:IFA$="" THEN 3080
3090 IF A$=CHR$(13) THEN RETURN
3100 PRINT "♥":GOTO 3000
3500 PRINT "♥";TAB(5);
   "■TEMPERATURE":PRINT
   "■CONVERT -"
3505 PRINT "■CENTIGRADE TO":
   PRINT "FAHRENHEIT (C)"
3506 PRINT "■FAHRENHEIT TO":
```




```
    PRINT"CENTIGRADE (F)"
3510 GET B$:IF B$<>"C" AND
    B$<>"F" THEN 3510
3520 IF B$="C" THEN 3560
3530 PRINT "■INPUT DEGREES":INPUT
    "FAHRENHEIT";VL
3540 PRINT "♥";VL;"DEGREES":PRINT
    "FAHRENHEIT EQUALS"
3550 PRINT "■";(VL-32)*5/9:PRINT
    "DEGREES CENTIGRADE":
    GOTO 3590
3560 PRINT "■INPUT DEGREES":INPUT
    "CENTIGRADE";VL
3570 PRINT "♥";VL;"DEGREES":PRINT
    "CENTIGRADE EQUALS"
3580 PRINT "■";32+VL*9/5:PRINT
    "DEGREES FAHRENHEIT"
3590 GET A$:IF A$="" THEN 3590
3600 IF A$=CHR$(13) THEN RETURN
3610 PRINT "♥":GOTO 3500
```

```
5 MODE6
10 DIML(7),L$(7),A(6),A$(6),V(6),
   V$(6),M(5),M$(5),P(4),P$(4)
20 FORK=ØTO7:READ L(K),L$(K):NEXT
30 DATA 1,INCHES,12,FEET,36,YARDS,
   63360,MILES,.03937,
   MILLIMETRES,.3937,CENTIMETRES,
   39.37,METRES,39370,KILOMETRES
40 FORK=ØTO6:READ A(K),A$(K):NEXT
50 DATA 1,SQ INCHES,144,SQ FEET,
   6272640,ACRES,4.0145E9,SQ
   MILES,.155,SQ CMS, 1550,SQ
   METRES,1.55E7,HECTARES
60 FORK=ØTO6:READV(K),V$(K):NEXT
70 DATA 1,CUBIC INCHES,1728,CUBIC
   FEET,34.67,PINTS,277.36,
   GALLONS,.06102,CC'S,61.024,
   LITRES,61024,CUBIC METRES
80 FORK=ØTO5:READM(K),M$(K):NEXT
90 DATA 1,OUNCES,16,POUNDS,35840,
   TONS,.03527,GRAMS,35.27,KILOGRAMS
   ,35270,TONNES
100 FORK=ØTO4:READP(K),P$(K):NEXT
110 DATA 1,PSI,51.73,mmHG,6895,
   N/SQ METRE,.0681,
   ATMOSPHERES,68.95,MILLIBARS
120 CLS:PRINT"WHICH CATEGORY (Ø-6) ?"
130 PRINTTAB(10,6)"Ø) QUIT PROGRAM"
   TAB(10,8)"1)□LENGTH"TAB(10,10)
   "2)□AREA"TAB(10,12)
   "3)□VOLUME"TAB(10,14)
   "4)□WEIGHT"TAB(10,16)
   "5)□PRESSURE"TAB(10,18)
   "6)□TEMPERATURE"
   140 A$=GET$:IF A$<"Ø" OR A$>"6"
   THEN 140
150 IF A$="Ø" THEN CLS:END
160 CLS:ON VAL(A$) GOSUB 1000,1500,
   2000,2500,3000,3500
```

```
170 GOTO 12Ø
1ØØØ PRINT'"LENGTH"'"SELECT ORIGINAL
   UNITS":FOR K=Ø TO 7:PRINT
   TAB(1Ø,6+2*K);K+1;")□";L$(K):
   NEXT
1Ø1Ø B$=GET$:IF B$<"1" OR B$>"8"
   THEN 1Ø1Ø
1Ø2Ø B=VAL(B$)-1:CLS:PRINT"INPUT
   NUMBER OF□";L$(B)
1Ø3Ø INPUT VL
1Ø4Ø CLS:PRINT'''VL;"□"L$(B);
   "□EQUALS"
1Ø5Ø IF B>3 THEN 1Ø8Ø
1Ø6Ø FORK=ØTO3:PRINTTAB(5,1Ø+K*2)
   VL*L(B)/L(K+4)TAB(25,1Ø+K*2)
   L$(K+4):NEXT
1Ø7Ø GOTO 1Ø9Ø
1Ø8Ø FORK=ØTO3:PRINTTAB(5,1Ø+K*2)
   VL*L(B)/L(K)TAB(25,1Ø+K*2)
   L$(K):NEXT
1Ø9Ø A$=GET$
11ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 1ØØØ
15ØØ PRINT'"AREA"'"SELECT ORIGINAL
   UNITS":FOR K=Ø TO 6:PRINT
   TAB(1Ø,6+2*K);K+1;")□";A$(K):
   NEXT
151Ø B$=GET$:IF B$<"1" OR B$>"7"
   THEN 151Ø
152Ø B=VAL(B$)-1:CLS:PRINT"□INPUT
   THE NUMBER OF□";A$(B)
153Ø INPUT VL
154Ø CLS:PRINT'''VL;"□"A$(B);
   "□EQUALS"
155Ø IF B>3 THEN 158Ø
156Ø FOR K=ØTO2:PRINTTAB(5,1Ø+K*2);
   VL*A(B)/A(K+4)TAB(25,1Ø+K*2)
   A$(K+4):NEXT
157Ø GOTO 159Ø
158Ø FOR K=Ø TO 3:PRINTTAB(5,1Ø+
   K*2);VL*A(B)/A(K)TAB(25,
   1Ø+K*2)A$(K):NEXT
159Ø A$=GET$
16ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 15ØØ
2ØØØ PRINT'"VOLUME"'"SELECT ORIGINAL
   UNITS":FOR K=Ø TO 6:PRINT
   TAB(1Ø,6+2*K);K+1;")□";V$(K):
   NEXT
2Ø1Ø B$=GET$:IF B$<"1"ORB$>"7"
   THEN 2Ø1Ø
2Ø2Ø B=VAL(B$)-1:CLS:PRINT"□INPUT
   NUMBER OF□";V$(B)
2Ø3Ø INPUT VL
2Ø4Ø CLS:PRINT'''VL;"□"V$(B);
   "□EQUALS"
2Ø5Ø IF B>3 THEN 2Ø8Ø
2Ø6Ø FORK=ØTO2:PRINTTAB(5,1Ø+K*2);
   VL*V(B)/V(K+4)TAB(25,1Ø+K*2)
   V$(K+4):NEXT
2Ø7Ø GOTO 2Ø9Ø
2Ø8Ø FORK=ØTO3:PRINTTAB(5,1Ø+K*2);
   VL*V(B)/V(K)TAB(25,1Ø+K*2)
   V$(K):NEXT
2Ø9Ø A$=GET$
21ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 2ØØØ
25ØØ PRINT'"WEIGHT"'"SELECT ORIGINAL
   UNITS":FOR K=Ø TO 5:PRINTTAB(1Ø,6
   +2*K);K+1;")□";M$(K):NEXT
251Ø B$=GET$:IF B$<"1" OR B$>"6"
   THEN 251Ø
252Ø B=VAL(B$)-1:CLS:PRINT"□INPUT
   NUMBER OF□";M$(B)
253Ø INPUT VL
254Ø CLS:PRINT'''VL;"□"M$(B);
   "□EQUALS"
255Ø IF B>2 THEN 258Ø
256Ø FOR K=Ø TO 2:PRINTTAB(5,1Ø+
   K*2);VL*M(B)/M(K+3)TAB(25,
   1Ø+K*2)M$(K+3):NEXT
257Ø GOTO 259Ø
258Ø FOR K=Ø TO 2:PRINTTAB(5,1Ø+
   K*2);VL*M(B)/M(K)TAB(25,
   1Ø+K*2)M$(K):NEXT
259Ø A$=GET$
26ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 25ØØ
3ØØØ PRINT'"PRESSURE"'"SELECT
   ORIGINAL UNITS":FOR K=Ø TO 4:PRINT
   TAB(1Ø,6+2*K);K+1;")□";
   P$(K):NEXT
3Ø1Ø B$=GET$:IF B$<"1" OR B$>"5"
   THEN 3Ø1Ø
3Ø2Ø B=VAL(B$)-1:CLS:PRINT"□INPUT
   NUMBER OF□";P$(B)
3Ø3Ø INPUT VL
3Ø4Ø CLS:PRINT'''VL;"□"P$(B);
   "□EQUALS"
3Ø5Ø T=Ø:FOR K=Ø TO 4:IF K=B THEN
   3Ø7Ø
3Ø6Ø PRINTTAB(5,1Ø+K*2);VL*P(K)/
   P(B)TAB(25,1Ø+K*2)P$(K)
3Ø7Ø NEXT
3Ø8Ø A$=GET$
31ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 3ØØØ
35ØØ PRINT'"TEMPERATURE"'"CONVERT
   CENTIGRADE TO FAHRENHEIT (C) OR
   FAHRENHEIT TO CENTIGRADE (F)"
351Ø B$=GET$:IF B$="C" THEN 356Ø
352Ø IF B$<>"F" THEN 351Ø
353Ø INPUT"INPUT DEGREES
   FAHRENHEIT□",VL
354Ø CLS:PRINT';VL;"□DEGREES
   FAHRENHEIT EQUALS"
355Ø PRINT;(VL-32)*5/9;"□DEGREES
   CENTIGRADE":GOTO 359Ø
356Ø INPUT"INPUT DEGREES
   CENTIGRADE□",VL
357Ø CLS:PRINT';VL;"□DEGREES
   CENTIGRADE EQUALS"
358Ø PRINT;32+VL*9/5;"□DEGREES
   FAHRENHEIT"
359Ø A$=GET$
36ØØ IF A$=CHR$(13) THEN RETURN ELSE
   CLS:GOTO 35ØØ
```

```
1Ø DIML(7),L$(7),A(6),A$(6),V(6),
   V$(6),M(5),M$(5),P(4),P$(4)
2Ø FORK=ØTO7:READ L(K),L$(K):NEXT
3Ø DATA 1,INCHES,12,FEET,36,YARDS,
   6336Ø,MILES,.Ø3937,
   MILLIMETRES,.3937,
   CENTIMETRES,39.37,METRES,
   3937Ø,KILOMETRES
4Ø FORK=ØTO6:READA(K),A$(K):NEXT
5Ø DATA 1,SQ INCHES,144,SQ FEET,
   627264Ø,ACRES,4.Ø145E9,SQ
   MILES,.155,SQ CMS,155Ø,SQ
   METRES,1.55E7,HECTARES
6Ø FOR K=ØTO6:READV(K),V$
   (K):NEXT
7Ø DATA 1,CUBIC INCHES,1728,CUBIC
   FEET,34.67,PINTS,277.36,
   GALLONS,.Ø61Ø2,CC'S,61.Ø24,
   LITRES,61Ø24,CUBIC METRES
8Ø FORK=ØTO5:READM(K),M$(K):NEXT
9Ø DATA 1,OUNCES,16,POUNDS,3584Ø,
   TONS,.Ø3527,GRAMS,35.27,
   KILOGRAMS,3527Ø,TONNES
1ØØ FORK=ØTO4:READP(K),P$(K):NEXT
11Ø DATA 1,PSI,51.73,mmHG,6895,N/SQ
   METRE,.Ø681,ATMOSPHERES,
   68.95,MILLIBARS
12Ø CLS:PRINT"□□□WHICH CATEGORY
   (Ø-6) ?"
13Ø PRINT@72,"Ø-□QUIT PROGRAM":
   PRINT@136,"1-□LENGTH":
   PRINT@2ØØ,"2-□AREA":
   PRINT@264,"3-□VOLUME":
   PRINT@328,"4-□WEIGHT":
   PRINT@392,"5-□PRESSURE":
   PRINT@456,"6-□TEMPERATURE"
14Ø A$=INKEY$:IF A$<"Ø" OR A$>"6"
   THEN 14Ø
15Ø IF A$="Ø" THENCLS:END
16Ø CLS:ON VAL(A$) GOSUB1ØØØ,15ØØ,
   2ØØØ,25ØØ,3ØØØ,35ØØ
17Ø GOTO 12Ø
1ØØØ PRINT@12,"length":PRINT@37,
   "SELECT ORIGINAL UNITS":FOR
   K=ØTO7:PRINT@136+32*K,K+1;
   "-□";L$(K):NEXT
1Ø1Ø B$=INKEY$:IF B$<"1" OR B$>"8"
   THEN 1Ø1Ø
1Ø2Ø B=VAL(B$)-1:CLS:PRINT"□INPUT
   NUMBER OF□";L$(B)
1Ø3Ø INPUT VL
1Ø4Ø CLS:PRINTVL;L$(B);
   "□EQUALS□"
1Ø5Ø IF B>3 THEN 1Ø8Ø
```

```
1Ø6Ø FORK=ØTO3:PRINT@96+K*64,
   VL*L(B)/L(K+4),L$(K+4):NEXT
1Ø7Ø GOTO 1Ø9Ø
1Ø8Ø FORK=ØTO3:PRINT@96+K*64,
   VL*L(B)/L(K),L$(K):NEXT
1Ø9Ø A$=INKEY$:IF A$="" THEN 1Ø9Ø
11ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO1ØØØ
15ØØ PRINT@13,"area":PRINT@37,
   "SELECT ORIGINAL UNITS":FOR
   K=ØTO6:PRINT@136+32*K,K+1;"—
   □";A$(K):NEXT
151Ø B$=INKEY$:IF B$<"1"ORB$>"7"
   THEN 151Ø
152Ø B=VAL(B$)—1:CLS:PRINT"□INPUT
   THE NUMBER OF□";A$(B)
153Ø INPUT VL
154Ø CLS:PRINTVL;A$(B);"□EQUALS"
155Ø IF B>3 THEN 158Ø
156Ø FORK=ØTO2:PRINT@96+K*64,
   VL*A(B)/A(K+4),A$(K+4):NEXT
157Ø GOTO 159Ø
158Ø FORK=ØTO3:PRINT@96+K*64,VL*
   A(B)/A(K),A$(K):NEXT
159Ø A$=INKEY$:IF A$="" THEN 159Ø
16ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 15ØØ
2ØØØ PRINT@12,"volume":PRINT@37,
   "SELECT ORIGINAL UNITS":FOR
   K=ØTO6:PRINT@136+32*K,K+1;
   "—□";V$(K):NEXT
2Ø1Ø B$=INKEY$:IFB$<"1"ORB$>"7"
   THEN 2Ø1Ø
2Ø2Ø B=VAL(B$)—1:CLS:PRINT"□INPUT
   NUMBER OF□";V$(B)
2Ø3Ø INPUT VL
2Ø4Ø CLS:PRINTVL;V$(B);"□EQUALS□"
2Ø5Ø IF B>3 THEN 2Ø8Ø
2Ø6Ø FORK=ØTO2:PRINT@96+K*64,
   VL*V(B)/V(K+4),V$(K+4):NEXT
2Ø7Ø GOTO2Ø9Ø
2Ø8Ø FORK=ØTO3:PRINT@96+K*64,
   VL*V(B)/V(K),V$(K):NEXT
2Ø9Ø A$=INKEY$:IF A$="" THEN 2Ø9Ø
21ØØ IF A$=CHR$(13)THEN RETURN
   ELSECLS:GOTO 2ØØØ
25ØØ PRINT@13,"weight":PRINT@37,
   "SELECT ORIGINAL UNITS":FOR
   K=ØTO5:PRINT@136+32*K,K+1;
   "—□";M$(K):NEXT
251Ø B$=INKEY$:IF B$<"1"ORB$>"6"
   THEN 251Ø
252Ø B=VAL(B$)—1:CLS:PRINT"□INPUT
   NUMBER OF□";M$(B)
253Ø INPUT VL
254Ø CLS:PRINTVL;M$(B);"□EQUALS"
255Ø IF B>2 THEN 258Ø
256Ø FORK=ØTO2:PRINT@96+K*64,
   VL*M(B)/M(K+3),M$(K+3):NEXT
257Ø GOTO 259Ø
258Ø FORK=ØTO2:PRINT@96+K*64,
   VL*M(B)/M(K),M$(K):NEXT
259Ø A$=INKEY$:IF A$="" THEN 259Ø
26ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO25ØØ
3ØØØ PRINT@11,"pressure":PRINT@37,
   "SELECT ORIGINAL UNITS":FOR
   K=ØTO4:PRINT@136+32*K,K+1;
   "—□";P$(K):NEXT
3Ø1Ø B$=INKEY$:IF B$<"1"ORB$>"5"
   THEN 3Ø1Ø
3Ø2Ø B=VAL(B$)—1:CLS:PRINT"□INPUT
   NUMBER OF□";P$(B)
3Ø3Ø INPUT VL
3Ø4Ø CLS:PRINTVL;P$(B);"□EQUALS"
3Ø5Ø T=Ø:FORK=ØTO4:IF K=B THEN 3Ø7Ø
3Ø6Ø PRINT@96+T*64,VL*P(K)/P(B),
   P$(K):T=T+1
3Ø7Ø NEXT
3Ø8Ø A$=INKEY$:IF A$="" THEN 3Ø8Ø
3Ø9Ø IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO3ØØØ
35ØØ CLS:PRINT@1Ø,"temperature":
   PRINT@37,"CONVERT —":
   PRINT"□CENTIGRADE TO FAHRENHEIT
   (C) OR FAHRENHEIT TO CENTIGRADE
   (F)"
351Ø B$=INKEY$:IFB$<>"C" AND
   B$<>"F" THEN 351Ø
352Ø IF B$="C" THEN 356Ø
353Ø PRINT"□INPUT DEGREES
   FAHRENHEIT":INPUT VL
354Ø CLS:PRINT@33,VL;"DEGREES
   FAHRENHEIT EQUALS"
355Ø PRINT@97,(VL—32)*5/9;
   "DEGREES CENTIGRADE":
   GOTO 359Ø
356Ø PRINT"□INPUT DEGREES
   CENTIGRADE":INPUT VL
357Ø CLS:PRINT@33,VL;"DEGREES
   CENTIGRADE EQUALS"
358Ø PRINT@97,32+VL*9/5;
   "DEGREES FAHRENHEIT"
359Ø A$=INKEY$:IF A$=
   "" THEN 359Ø
36ØØ IF A$=CHR$(13) THEN RETURN
   ELSECLS:GOTO 35ØØ
```



PICTURES FROM UDGS-2

Here are two more characters—a snake and a monkey—to complete your jungle scene, plus some ideas on how to animate the picture and how to save memory

In earlier parts of this feature, you have already seen how to define large numbers of characters on your computer, and started to use this to build a picture of a jungle scene.

The jungle scene program was made up of a number of sections, each one defining the UDGs (except on the Dragon and Tandy, see page 489). Each section made part of the picture—the crocodile, elephant, trees and so on. The second half of the program, which follows, adds a snake and a monkey and finishes the background.

If you SAVEd the last half on tape or disc you can LOAD it back again, as this program needs the first half to RUN. If you have an Acorn computer, type this line before loading in the old program, as you did last time. Then type NEW and RETURN.

```
PAGE=PAGE+&600
```

The additional data lines you need to define the images are printed after the main programming. Type these lines in first:

```
130 REM poke snake data
140 POKE 23676,254
150 FOR n=USR "a" TO USR "r"+7: READ
    a: POKE n,a: NEXT n
160 POKE 23676,253
170 FOR n=USR "a" TO USR "j"+7: READ
    a: POKE n,a: NEXT n
180 REM poke monkey data
190 POKE 23676,252
200 FOR n=USR "a" TO USR "u"+7:
    READ a: POKE n,a: NEXT n
210 POKE 23676,251
220 FOR n=USR "a" TO USR"“u"+7:
    READ a: POKE n,a:NEXT n
230 POKE 23676,250
240 FOR n=USR "a" TO USR "g"+7:
    READ a: POKE n,a: NEXT n
500 REM print snake
510 INK 1
520 POKE 23676,254
530 LET z=144: FOR n=2 TO 7: FOR m=16
    TO 18: PRINT AT m,n;CHR$ z: LET
    z=z+1: NEXT m: NEXT n
540 POKE 23676,253
550 LET z=144: FOR n=8 TO 12: FOR
    m=17 TO 18: PRINT AT m,n;CHR$ z: LET
    z=z+1: NEXT m: NEXT n
600 REM print monkey
610 INK 0
620 POKE 23676,252
630 PRINT AT 0,30;CHR$ 144;AT 1,24;CHR$
    145;CHR$ 146;AT 1,28;CHR$ 147;CHR$
    148
640 PRINT AT 2,23;CHR$ 149;CHR$ 150;
    CHR$ 151;CHR$ 32;CHR$ 152;CHR$ 153
650 PRINT AT 3,23;CHR$ 154;CHR$
    155;CHR$ 32;CHR$ 32;CHR$ 156;CHR$
    157;CHR$ 158
660 PRINT AT 4,24;CHR$ 159;CHR$
    160;CHR$ 32;CHR$ 161;CHR$ 162;CHR$
    163;CHR$ 164
670 POKE 23676,251
680 PRINT AT 5,24;: FOR n=144 TO 151:
    PRINT CHR$ n;: NEXT n
690 PRINT AT 6,24;CHR$ 152;CHR$
    153;CHR$ 153;CHR$ 153;CHR$ 153;CHR$
    154;CHR$ 155;CHR$ 156
700 PRINT AT 7,23;CHR$ 157;CHR$
    158;CHR$ 159;CHR$ 32;CHR$ 32;CHR$
    160;CHR$ 161;CHR$ 162;AT 8,22;CHR$
    163;CHR$ 164;
710 POKE 23676,250
720 PRINT CHR$ 144;CHR$ 145;CHR$
    32;CHR$ 32;CHR$ 146;CHR$ 147;AT 9,24;
    CHR$ 148;CHR$ 149;AT 10,24;
    CHR$ 150
850 REM sun
855 INK 6
860 FOR n=0 TO 2*PI STEP .05: PLOT
    70,150: DRAW SIN n*12,COS n*12:
    NEXTn
870 FOR n=0 TO 2*PI STEP PI/4: PLOT
    70,150: DRAW COS n*20,SIN n*20:
    NEXTn
970 INK 0
```

The additional data lines you need to define the images are printed after the main programming.

```
51 FOR Z=144 TO 759: READ X:
   POKE 13312+Z,X:NEXT Z
100 M$="■■■■/0■■
    12○.■■■■■■■■■■
    ■■■■■■■"
110 M$=M$+"■■345■67
    ■■■■■■■■■■■89■■:;<■
    ■■■■■■■■■■=>■?■■■
    ■■■■■■■■■■■■■■■■■
    ■■■■■■■■■■■■■■■■
    ■■"
120 M$=M$+"■■■■■■■
    ■■■■■■■■■■■■■■■■
    ■■●■■■■■■■■■■■■
    ■■■■■■■■■■○■■■■■
    ■■■■■■■■■■■■■■■
    ■π■"
140 PRINT"■■■■■■■
    ■■RUX[↑!"C$"■■■■■
    SVY£←"CHR$(34)"S&(*,"C$
    "■■■■■TWZ]□#%')+-"
155 POKE 1088,81
175 TT=T:PRINT"■"TAB(T)"■"
    M$:FORD=1TO50:POKE55360,
    RND(1)*7+1:NEXTD
180 PRINT"■"TAB(T)"■"M$
```

The additional data lines you need to define the images are printed after the main program.

```
180 PROCSNAKE(128,100,146)
190 PROCMONKEY(770,1023,174)
210 COLOUR3:COLOUR128
```



- ADDING DATA TO COMPLETE THE JUNGLE SCENE
- DRAWING A SNAKE AND A MONKEY
- COMPLETING THE BACKGROUND
- USING HIGH RESOLUTION GRAPHICS
- ANIMATING THE CHARACTERS
- **SAVE**ING THE DATA AS MACHINE CODE



```
220 VDU 31,35,4:PRINT"HI"
230 VDU 31,34,5:PRINT"THERE"
410 DEF PROCSNAKE(X,Y,Z)
420 VDU 5
430 GCOLØ,Ø
440 FOR T=Ø TO 17
460 MOVE X+32*(T DIV 3),Y-32*(T MOD
   3):VDU T+Z
470 NEXT
480 FOR T=18 TO 27
490 MOVE X+32*(-3+(T DIV
   2)),Y-32*((T MOD 2)+1):VDU T+Z
500 NEXT
510 VDU 4
520 ENDPROC
530 DEF PROCMONKEY(X,Y,Z)
540 VDU 5
550 GCOLØ,3
560 RESTORE 196Ø
570 FOR T=Ø TO 48
580 READ A
590 MOVE 32*(A MOD 1Ø)+X,
   Y-32*(A DIV 1Ø):VDU Z+T
600 IF A=63 THEN VDU Z+T,Z+T,Z+T
610 NEXT
620 VDU 4
630 ENDPROC
1160 MOVE42Ø,8ØØ
1170 GCOLØ,3
1180 FOR T=Ø TO PI*2+.Ø8 STEP
   .Ø8:DRAW 3ØØ+12Ø*COS(T),
   8ØØ+1ØØ*SIN(T):NEXT
1190 FOR T=7ØØ TO 896:PLOT 77,
   3ØØ,T:NEXT
1200 FOR T=Ø TO PI*2 STEP .3:
   MOVE3ØØ,8ØØ:DRAW 3ØØ+5ØØ
   *COS(T),8ØØ+45Ø*SIN(T):NEXT
```

The Dragon and Tandy jungle



```
1960 DATA 8,12,13,16,17,21,22,23,25,
    26,31,32,35,36,37,42,43,45,46,47,
    48,52,53,54,55,56,57,58,59,62,63,
    67,68,69,71,72,73,76,77,78,80,81,
    82,83,86,87,92,93,102
```

Owners of a BBC with operating system 0.1 should delete Line 1190 and change Line 1180 to:

```
1180 FOR T=0 TO PI*2+.08 STEP
    .08: MOVE 300,800: PLOT85,300
    +120*COS(T), 800+100*SIN(T):
    NEXT
```

Type in these data lines to define the shape of the UDGs.

```
1400 REM SNAKE
1410 DATA 0,0,0,0,0,0,3,14,31,31,63,67,
    65,254,254,225,71,63,0,0,0,0,0,0
1420 DATA 0,0,0,0,255,12,28,28,28,254,
    143,1,0,4,248,208,224
1430 DATA 0,0,0,0,0,0,0,0,0,0,128,240,56
1440 DATA 52,99,225,225,225,96,47,31,14,
    6,2,1,0,0,0,0,0,0,0,3,15,63,127
1450 DATA 255,225,240,248,252,255,
    255,126,60,28,124,247,247,115,
    31,1,0,0,0,0
1460 DATA 255,191,159,143,199,194,
    127,64,128,128,64,63
1470 DATA 48,112,248,249,253,255,189,255,
    0,0,0,0,192,252,190,63,63,24,240
1480 DATA 9,5,7,5,249,67,243,251,255,252,
    238,239,255
1490 DATA 192,224,208,223,224,129,
    193,227,243,247,255,248,240,224,192
1500 DATA 0,0,0,0,255,129,195,199,199
1510 DATA 239,239,255,0,0,0,0,0,0,0,0,
    128,248,31,140,158,191,191,255,
    0,0,0,0
1520 DATA 0,0,0,1,15,62,206,30,61,121,
    243,247,247,63,15,3,0
1530 DATA 30,127,191,95,93,61,250,240,
    194,252,194,192,192,224,224,240
1540 REM MONKEY
1550 DATA 1,2,4,8,16,32,64,128,0,0,0,3,
    31,63,127,252,0,0,0,224,248,252,252
1560 DATA 62,3,15,15,15,15,14,14,12
1570 DATA 3,142,240,192,128,0,0,0,1,1,3,
    3,3,3,3,3,248,240,224,224,192
1580 DATA 192,192,192,30,14,110,76,56,
    0,0,0,0,0,0,0,0,0,0
1590 DATA 1,28,28,56,56,112,112,224,224
1600 DATA 3,3,1,1,1,0,0,0,192,224,224,
    240,248,248,252,126,3,7,15,31,63,62
1610 DATA 126,252,192,192,128,0,0,0,3,7,
    0,0,0,0,0,0,128,223
1620 DATA 127,63,31,31,15,7,7,3,0,128,128,
    192,192,192,224,240,252,126,126,127
1630 DATA 63,63,31,31,15,15,7,3,3,131,
    131,131
```



```
1640 DATA 255,255,231,231,255,252,
    172,183,128,192,224,224,224,
    224,224,224
1650 DATA 1,1,0,0,1,3,7,248,255,255,
    255,255,255,255,255,0
1660 DATA 255,255,255,255,255,255,255
1670 DATA 15,255,255,255,255,255,255,255,
    199,255,255,255,255,255,255,255
1680 DATA 223,239,255,255,252,128,0,128,
    224,240,248,248,240,96,0,0,0,0,200
1690 DATA 232,248,248,120,112
1700 DATA 7,15,15,31,31,31,63,63,255,255,
    255,255,255,255,255,255,192,224,224
1710 DATA 224,224,224,192,192,0,0,1,1,3,
    7,14,60,240,224,192,128,128,0,0,0
1720 DATA 0,0,0,0,0,3,15,31,63,63,63,63,
    126,254,252,240,191,191,63,63,63
1730 DATA 63,63,63,15,7,7,7,7,7,7,7,192,
    131,135,159,191,255,254,252
1740 DATA 252,248,240,224,192,128,0,0,0,
    1,3,7,6,6,4,0,127,254,240,224
1750 DATA 64,64,0,0,192,0,0,0,0,1,1,3,63,
    31,63,127,254,252,248,240
1760 DATA 7,3,0,0,0,0,0,0,240,192,0,0,0,
    0,0,0,7,7,15,31,31,31,23,23,224
1770 DATA 192,128,0,0,0,0,0,7,6,4,0,0,0,0,0
```

```
90 PCLS3
100 DRAW"BM62,1C2DG6LG2LGLUHLGLD
    6GDGDGDG7DG2DRD3RD2FDFDL12H
    2LULU3HUH4UH2ULU5EUE4R3F3D
    2GLHURBD2R2E2U3HUHLHL4GL2G
    4DGD7FD2FDFDF2D"
```



```
110 DRAW"F2DFD2G3DGDGD2GD5G2L2GL
    G2LGLG2D3EUER2ND2RE2R2ER2ER
    EREUEUEUERD9FG4DG2DG2D2ND
    2R2D5E2U5E8U9E3R17F3D6FR3ER
    ERE2RE5UE3UE3UE"
120 DRAW"U2NU2LH2LD3FDGDGDG4LGL
    GLG2LG2HU2EUEU4H3E2R4ERFRFE
    URUH2U6H3L4GH2L2G2DF2D4G
    2L2H2U2HUHUHUEUEUE2UEUEUE
    UEU2EUEREUL4D5L3D6L2D3L2D3"
130 DRAW"BM24,16U3NL6DL8D2L2D2
    L2D9BM7Ø,43D4G8BM3,64E3BM
    +1Ø,1ØD3"
14Ø PAINT(5Ø,5Ø),2
15Ø DRAW"BM56,34C1RDBF3DBL7DF
    2BE2BU2"
16Ø GET(Ø,Ø)—(76,8Ø),M,G
17Ø PCLS
18Ø DRAW"BM13,11C4G3L8H2U2EUE2E
    RERER8FR2F3RE3RERER11FR3F2RF
    4R12FR3FR4ERER2ERERER3FD
    4GBU3GDG2LGFNR3GD2FDFL4"
19Ø DRAW"H2LHLHLHL18G4LGL16HL
    3HLH9LHL2HL2G3DFR3BM23,6D3FDF
    5RF2R9ER2ER7BM52,8LGL3G2L
    11H2UE2R7F4"
2ØØ PAINT(6Ø,1Ø),3,4:PAINT(13,5),3,4
21Ø DRAW"BM83,8C1DBL4BG5H3G4H
    5G5H4G5H3G4H4G3H8E4F2E2F2E2F
    2E2F4BM7,13LH3E6F4E4FDFD
    2F9R2FR15E"
22Ø GET(Ø,Ø)—(88,21),S,G
37Ø CIRCLE(5Ø,3Ø),25,2:PAINT
    (5Ø,3Ø),2:DRAW"BM5Ø,3ØC2NU
    3ØNE6ØNR84NF6ØND84NG6ØNL
    84NH3Ø"
43Ø PUT(1Ø,166)—(98,187),S,PSET
44Ø PUT(14Ø,Ø)—(216,8Ø),M,PSET
```

The additions to each computer fit in exactly with the last program, and just extend it to include more characters—although on the Acorn computers, Dragon and Tandy you need the lines given below to call up the new routines. Again, the DATA statements from Lines 14ØØ to 177Ø are common to the Spectrum, Commodore and Acorn computers. The Spectrum program starts off by adding more lines to POKE in the extra DATA, and the Acorn adds two new PROCedures.

The Commodore listing, too, is just an extension of the last program: Line 51 adds another FOR . . . NEXT loop to POKE in the extra DATA for the new characters. Lines 1ØØ to 12Ø create the monkey, Line 14Ø creates the snake and Line 155 places a moon on the screen. The UDGs for the monkey are put into the string M$ to save memory.

The Dragon draws the new characters, and GETs them into an array in the same way as it did for the characters in the last program.

Each program also prints the UDGs in the correct positions to fit in with the picture.

MORE CHANGES

In addition to the extra lines above, the Acorn, Dragon and Tandy programs need changes to certain other lines:

```
5Ø FOR T=128 TO 251
2ØØ PROCELE (8ØØ,46Ø,223)
1Ø9Ø PROCTREE2 (85Ø+RND(3ØØ),
     P+RND(2Ø)—3Ø,243)
11ØØ PROCTREE1 (9ØØ+RND(25Ø),
     P+RND(2Ø),236)
114Ø PROCTREE2 (RND(5ØØ),3ØØ
     +RND(5Ø),243)
```



The jungle scene on the Acorn

```
2Ø DIMC(59),M(156),S(48),E(17),
   T1(1),T2(7),F1(7),F2(7)
```

For the Acorn, these lines just change the first character used by the PROCedure, to compensate for the fact that there are now several more UDGs in memory. The new Line 5Ø changes the FOR . . . NEXT loop to define the new number of UDGs.

The Dragon and Tandy just DIMension two more arrays, one for each new character.

THE COMPLETE PICTURE

As you can see if you RUN the program, the screen is now full of characters and the picture seems complete. The snake, the monkey, and the sun (or, if you have a Commodore 64, the moon) are added by the extra lines.

You might equally well have added more elephants, trees or crocodiles to the picture, but in any case you now have a whole set of characters to use in your own pictures.

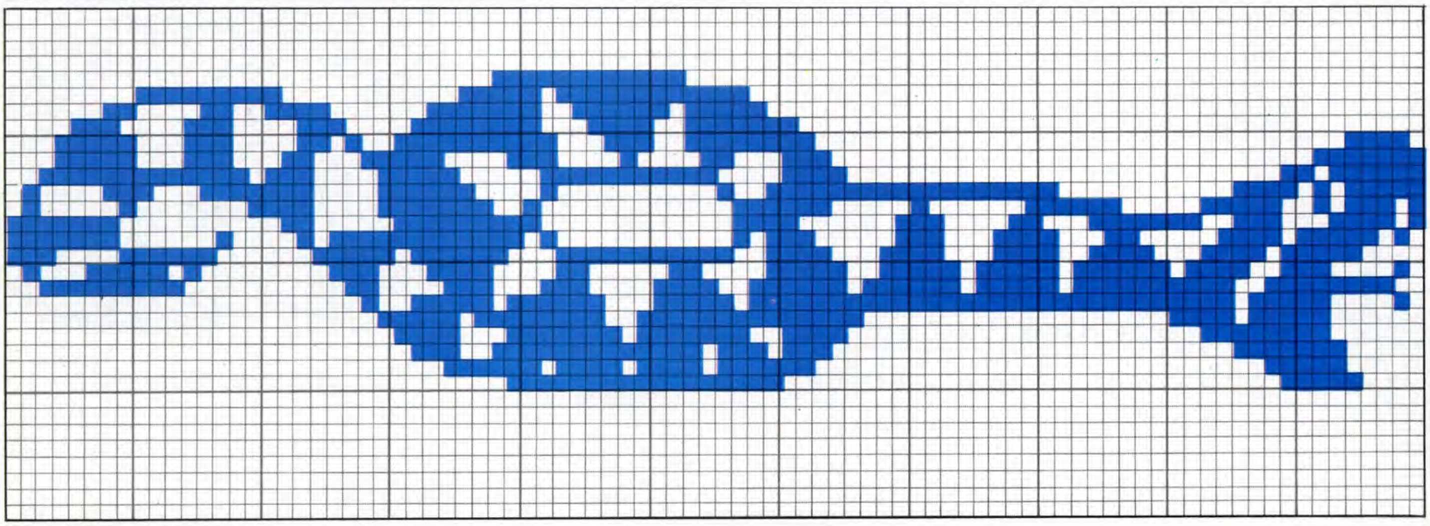
A HIGH RESOLUTION SUN

Note how high resolution graphics have been used as well as normal character graphics—there is no reason why you should not combine the two, and the mixture gives a surprisingly good result.

The Spectrum 'fills' its sun by drawing a lot of radii very close together, while the Dragon and Tandy are able to use their PAINT command to fill a circle. The Acorns use a suitable PLOT command to fill in the circle.

You can alter the picture further with other high resolution graphics: for example, you could add more hills, or even draw some lines to make the ground seem textured or cracked.

The Commodore does not have high resolution graphics commands in BASIC, and it is difficult to draw the large yellow sun of the other computers. So an ordinary graphics character is printed instead—in this case shaped like a moon.



NEXT STEPS

The range of changes you can make just with the UDGs is almost unlimited. Apart from changing the number of trees, snakes, monkeys and so on, you can also use the graphics characters which make up these pictures for other things.

An obvious example here, is to use the tree-tops in a different colour (white, if you can have white as well) for clouds, or as bushes.

You should watch out for colour clashes, though. Suppose you want to use the tree-tops as bushes; you would need to place the bushes at the top of each hill, since the hills are green, and so the bushes would not show up otherwise (unless you had red bushes, of course . . .).

ANIMATING THE PICTURE

The advantages of building your picture from UDGs do not just end with the number of ways you can rearrange it once it is finished: you can convert it into a moving picture with very few changes.

To animate the animals, for example, you could define one or two extra characters—a new trunk for the elephant, the snake with its head rearing up, or the monkey eating a banana are just a few of the many possibilities.

Whatever you decide to have the animals doing, once you have defined the extra UDGs, you can simply follow the procedure for animation outlined in the articles on pages 350 to 353, and pages 26 to 32.

If you have a Commodore, you can already see a fairly crude example of this: the monkey and the crocodile are moved by the last routine in the program. All this does is PRINT the characters in the background colour and then rePRINT them elsewhere. The moon also changes colour. You can turn off the animation by altering Line 19Ø to GOTO 19Ø.

By using a series of different UDGs for different positions, and then PRINTing each one in turn, the movement can be made much more realistic, and either amusing or frightening, or whatever else you wish it to be.

HOW THE UDGs FIT TOGETHER

If you are going to use the UDGs for animation, or in still pictures of your own, you can see how the various characters make up the monkey and the snake from the diagram above (the last article, on pages 484 to 491 shows how the crocodile is composed from the UDGs).

The microtip gives some possible replacement DATA for the elephant which you might like to use for the animation.

While the UDGs for the elephant, crocodile, and snake might seem restricted to a jungle or zoo, you can use the trees (and bushes or clouds) in almost any picture that you might want to have in your programs. You should be able to design just as impressive screen displays yourself using similar methods.

SAVEING MEMORY

The programs above use a lot of UDGs in order to show you how to use more than your computer's limit, and to give you some characters that you might like to use in your own pictures.

But you can produce an impressive picture with far less UDGs, if you need to, by using each UDG over and over again. For example, you could create a whole herd of elephants. Or, if you included a wall in your picture, it could cover a large proportion of the screen, and yet only need two UDGs!

By careful planning of what you want to draw, you can successfully create very interesting pictures with a surprisingly small number of UDGs. However, as it is quite straightforward to have large numbers of UDGs, the only advantages to be gained from 'economising' on

Microtip

Animate your characters

Here is some DATA which you can incorporate into the main program to animate the elephant. It simply gives an alternative trunk position, so you could switch between the two to simulate motion. The Dragon and Tandy have a DRAW command instead of DATA which you must GET into an array in the same way as the program does for the other characters.

```
DATA Ø,Ø,Ø,Ø,Ø,1,3,3
DATA 6,15,15,31,29,25,26,24
DATA 24,24,12,12,6,Ø,Ø,Ø
```

```
PCLS: DRAW"BM5,7C2GLFLD5F"
```

For each computer, you must also change the relevant FOR. . .NEXT loops to set up and print the new UDGs and you need to DIMension a new array for the Dragon and Tandy.

You can improve the animated elephant by moving it to a more prominent position than it is in at the moment.

Animation tends to emphasise a character, so it is a good idea to use the animated part of your picture as the principal part. You can then divert the user's or viewer's eyes from any weaker parts of the picture.

You can try animating any of the other characters in the same way.

UDGs are in terms of the time you save by not typing them in, and the memory taken up by the DATA. The Spectrum, in particular, uses up a lot of memory to store its DATA, but there are ways to make some savings.

You can conserve the computer's memory by SAVEing the bytes which make up your UDGs as a block of memory, and then deleting the DATA statements. Normally the computer stores each byte of DATA twice: once in the memory locations you POKE with the DATA, and again in the DATA statements themselves.

To SAVE the DATA as bytes of CODE, use:

```
SAVE "filename" CODE x,y
```

where x is the start address of the block of memory you want to SAVE, and y is the length of the block.

You should know the start address of the block of memory, as you need it to either POKE in the DATA, or to POKE the UDG pointer with the new start address. The article on pages 450 to 457 explains how to find out what it is.

The length of the block is quite simple: it is normally the number of UDGs you have in memory, times 8. In fact the program above which defines the UDGs for the jungle picture uses eight banks of UDGs, and starts each bank 256 bytes apart (instead of the minimum 168 bytes). The reason for this is that it lets you change the pointer with just one POKE, as opposed to the normal two. It also means that if you want to SAVE the bytes as CODE, you must SAVE a block (256*8) bytes long.

So, this command SAVEs the bytes for the jungle UDGs:

```
SAVE"jungle UDG" CODE 63488,2048
```

To LOAD the bytes back in again, you just type LOAD""CODE. To LOAD it back to an address other than where it was SAVEd from, just add the new start address after the LOAD""CODE.

The Commodore SAVEs machine code, or bytes of memory, to tape by altering four pointers to make the BASIC area correspond exactly with the block of memory you want to SAVE. It then SAVEs the data as normal.

For example, to SAVE the data from the UDGs in the program above, first type:

```
POKE 43,0: POKE44,48: POKE45,230:
POKE 46,55
```

and press RETURN. Now enter this:

```
SAVE"UDG DATA",1,1
```

The computer now SAVEs a block of memory which starts where your character data begins, and ends at the end of your data.

The ,1 immediately after the filename tells the computer that it is SAVEing to tape, while the second ,1 tells it that it is SAVEing machine code, or data.

The four POKEs change the addresses which point to the start and end of BASIC. The article on pages 450 to 457 explains how this works for the start of BASIC (for the two POKEs, 43 and 44) and again with the other two POKEs, 45 and 46, for the end of BASIC. To LOAD the data back again, use this:

```
LOAD"file name",1,1
```

The Acorn SAVEs data, or any block of memory, using the command *SAVE, followed by the file name and two numbers.

The first number is the first address in the block of memory you wish to SAVE, while the second number can be one of two things. It can either be the last address of the block of memory, plus one, or the length of the block of memory. If you are using the latter, you should put a plus sign at the beginning of the number: this tells the computer to add the number to the start address.

The data for the characters used by the programs above are stored in two places; one for the 'normal' UDGs, and another for the 'extra' characters. Here is the command to SAVE the first section of data; it uses the end address, plus one, as its second number.

```
*SAVE"UDGdata1" C00 D00
```

There are two points you should note from this: firstly that the numbers are in hexadecimal. The second point to watch is that each hex number is separated by a space.

To SAVE the second block of memory, first enter this line:

```
A%=131: PRINT~USR (&FFF4)
```

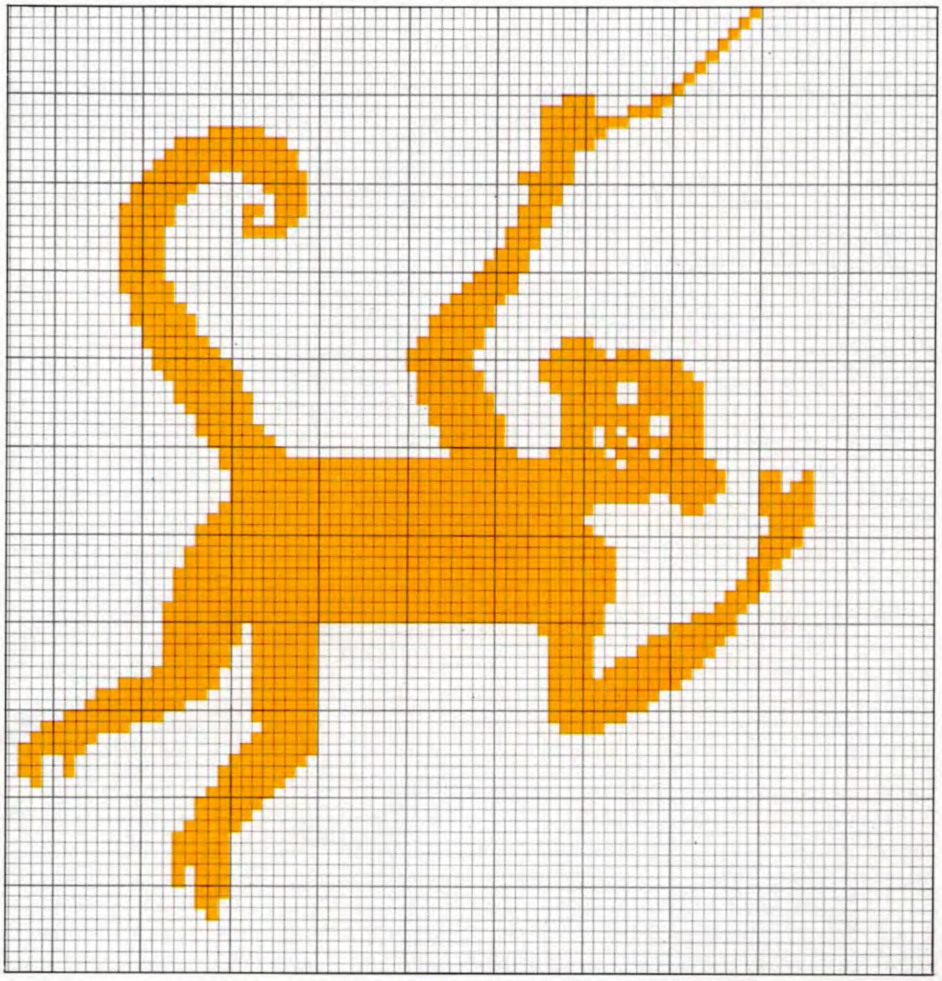
Take the middle four digits of the hex number that is PRINTed on the screen, and use it as the first number in this command:

```
SAVE"UDGdata2" (number) +600
```

You can LOAD it back in again using this:

```
*LOAD"file name"
```

These use the DRAW command, so there is no advantage in CSAVEing DATA.



WHEELING AND DEALING

Computers can be very good card players if you can program them correctly—and they never get bored! Here's how to program the graphics for a pack of cards

Ostracized by your friends, relatives and colleagues for milking them of their cash at cards? Penniless from playing experts? In either case, over the next three parts of Games Programming, you'll find the answer. Programming your computer to play Pontoon will give you a willing victim, or a way of playing without emptying the coffers.

In this first part you'll see how to set up the graphics routine that makes up your pack of cards. The remainder of the program—the game itself—will follow in two sections. But don't forget to SAVE each section on tape as you build up the game.

If you are not a Pontoon expert, don't worry. A full set of playing rules will be printed with the last part of the program. But first you need to program a pack of cards.

The graphics routine which will enable you to display cards on the screen looks like this:

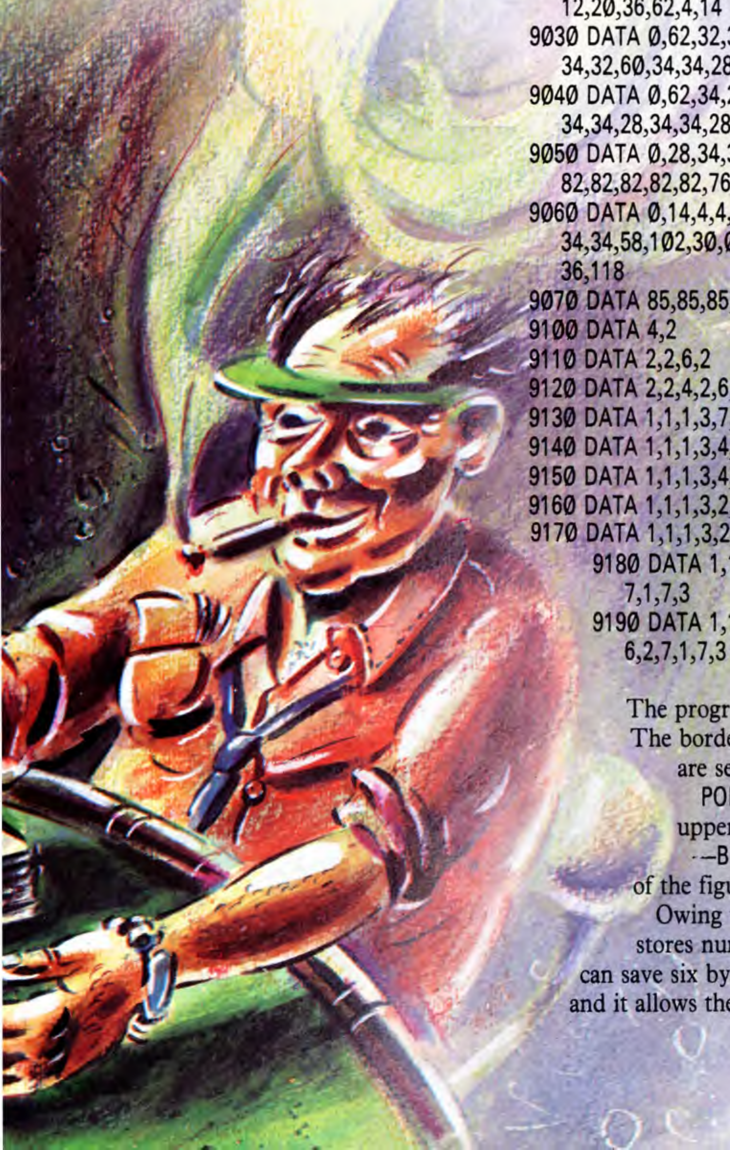
```
10 BORDER 4: PAPER 4: INK 9: CLS : POKE
   23658,8: LET B=Ø: LET C=1
20 FOR N=USR "A" TO USR "R"+7:
   READ A: POKE N,A: NEXT N
30 DIM C(52): FOR N=C TO 52: LET
   C(N)=N: NEXT N
40 DIM A(13,13,2)
50 FOR N=C TO 1Ø: FOR M=C TO N: READ
   A(N,M,C),A(N,M,2): NEXT M: NEXT N
60 FOR N=11 TO 13: LET A(N,C,C)=4: LET
   A(N,C,2)=2: NEXT N
70 LET CC=C:LET CP=1ØØ
80 GOSUB 5ØØØ
5ØØ LET Y=Ø: LET X=1
525 LET Z=C(CC)
53Ø GOSUB 55ØØ
54Ø STOP
5ØØØ CLS : PRINT AT 1Ø,1Ø;
   "SHUFFLING NOW"
```



- CARD LAYOUTS
- HIGH RESOLUTION GRAPHICS
- POSITIONING THE SPOTS
- DRAWING THE CARDS
- COMMODORE ROM GRAPHICS
- SHUFFLING THE PACK
- DEALING THE CARDS
- USING UDGS
- FOUR SUITS, THIRTEEN CARDS
- PICTURE AND ACE VALUES

```
5010 FOR N=C TO 100
5020 LET X=INT (RND*52)+C
5030 LET Y=INT (RND*52)+C
5040 LET Z=C(X): LET C(X)=C(Y): LET
   C(Y)=Z
5050 NEXT N
5060 CLS : RETURN
5500 FOR N=Y TO Y+8: PRINT PAPER 7;AT
   N,X;"□□□□□": NEXT N
5510 LET ST=INT ((Z-C)/13)
5520 LET CH=144+ST
5530 LET VA=Z-(13*ST)
5540 IF ST<2 THEN INK 2
5560 LET AC=147+VA
```

```
5600 PRINT PAPER 7;AT Y,X;CHR$ AC;AT
   Y,X+4;CHR$ AC;AT Y+8,X;CHR$ AC;AT
   Y+8,X+4;CHR$ AC
5610 FOR N=C TO VA: IF A(VA,N,C)<>B
   THEN PRINT PAPER 7;AT Y+A(VA,N,C),
   X+A(VA,N,2);CHR$ CH
5620 NEXT N
5890 INK 9
5900 RETURN
9000 DATA 0,54,127,127,127,62,28,8,0,
   8,28,62,127,62,28,8,8,28,62,127,
   127,62,8,28,8,28,28,107,127,107,8,28
9010 DATA 0,8,20,34,34,62,34,34,0,28,
   34,2,4,24,32,62
9020 DATA 0,28,34,2,12,2,34,28,0,4,
   12,20,36,62,4,14
9030 DATA 0,62,32,32,60,2,34,28,0,28,
   34,32,60,34,34,28
9040 DATA 0,62,34,2,4,8,16,16,0,28,
   34,34,28,34,34,28
9050 DATA 0,28,34,34,30,2,34,28,0,76,
   82,82,82,82,82,76
9060 DATA 0,14,4,4,4,4,36,24,0,28,34,
   34,34,58,102,30,0,118,36,40,48,40,
   36,118
9070 DATA 85,85,85,85,85,85,85,85
9100 DATA 4,2
9110 DATA 2,2,6,2
9120 DATA 2,2,4,2,6,2
9130 DATA 1,1,1,3,7,1,7,3
9140 DATA 1,1,1,3,4,2,7,1,7,3
9150 DATA 1,1,1,3,4,1,4,3,7,1,7,3
9160 DATA 1,1,1,3,2,2,4,1,4,3,7,1,7,3
9170 DATA 1,1,1,3,2,2,4,1,4,3,6,2,7,1,7,3
9180 DATA 1,1,1,3,3,1,3,3,4,2,5,1,5,3,
   7,1,7,3
9190 DATA 1,1,1,3,2,2,3,1,3,3,5,1,5,3,
   6,2,7,1,7,3
```



The program works like this:

The border, paper and ink colours are set up by Line 1Ø and the POKE puts the machine into upper case. The two variables —B and C—are used instead of the figures Ø and 1 throughout. Owing to the way the Spectrum stores numbers and variables, you can save six bytes of memory each time and it allows the program to RUN on the 16K Spectrum.

Line 2Ø sets up the UDGs for symbols used for the four suits, and UDGs for numbers and letters used in the corners of the cards The DATA for these is held in Lines 9ØØØ to 9Ø7Ø. Next, Line 3Ø sets up an unshuffled pack of cards. Array A, DIMensioned in Line 4Ø is filled with DATA from Lines 91ØØ to 919Ø which gives the coordinates of the spots on each card. Line 5Ø fills part of the array with the positions of the suit symbols on the number cards. Line 6Ø fills the remainder of the array with the position of the picture symbols. It would be possible to design proper pictures for the cards, but it would be rather tedious to enter all the data, and also difficult to fit in the 16K machine.

Line 7Ø sets CC to 1, and CP to 1ØØ. CC is the current card, and is the element in the card array which is being dealt with by the program. CP is the number of chips in the player's possession.

Line 8Ø calls the shuffling subroutine, starting at Line 5ØØØ. The screen is cleared, and the shuffling message is displayed by Line 5ØØØ itself. The shuffling is done by choosing two cards at random and swapping their positions. The FOR ... NEXT loop between Lines 5Ø1Ø sees that 100 swaps are done. The laws of chance mean that sometimes the same card will be chosen by Lines 5Ø2Ø and 5Ø3Ø and it will be swapped with itself. It doesn't really matter, though, because a very thorough shuffle will be given by 100 swaps. If you are not happy with swapping only 100 times, you can alter the value in Line 5Ø1Ø, but it'll soon take a ridiculously long time.

The subroutine ends at Line 5Ø6Ø which clears the screen and RETURNs.

Line 525 sets variable Z equal to the value of the current card in array C, before Line 53Ø calls the subroutine at Line 55ØØ. This subroutine is concerned with displaying the cards on the screen. Line 55ØØ displays the cards—only the white background at this stage, but the symbols and numbers will be displayed by the rest of the subroutine. Line 551Ø works out which suit the chosen card is in—each of the suits is given a number from zero to three. Line 552Ø works out which character string the UDG for that particular suit is stored in. Finally, you'll want to know

the number of the card within the suit—is it an ace, a two or perhaps a queen?

Before the suit symbols can be displayed on the card, the ink colour must be set. Line 554Ø sets the colour to red if the suit number is one or zero, black otherwise.

Line 56ØØ draws the card's number by picking out the appropriate UDG from those set up earlier. The suit symbols are displayed by Line 561Ø. The coordinates of each symbol are picked out from array A, and the UDG is printed at that position.

Finally, INK 9—the contrast ink—is set, and the program RETURNs.



The program below is the Commodore graphics routine. If you have a Vic 20, make sure that you change Line 1Ø so that it reads as follows:

```
1Ø POKE 36879,3Ø:MU=1ØØ:JM=3:
   LE=21:TX=6
```

Commodore owners are very lucky because all the symbols needed for these card graphics are already on the machine, so special graphics routines do not have to be written:

```
1Ø POKE 5328Ø,6:POKE 53281,1:
   MU=1ØØ:JM=7:LE=39:TX=15
2Ø FOR Z=1 TO LE:E1$=E1$+"_":
   E2$=E2$+"▒":NEXT Z:
   E2$=E2$+"▒"
3Ø DIM D$(52):C$="♠♥♣♦":
   CC$="A23456789ØJQK"
4Ø FOR ZZ=1 TO 4:FOR Z=1 TO 13
5Ø X=X+1:D$(X)=MID$(C$,ZZ,1)
   +MID$(CC$,Z,1):NEXT Z,ZZ:PRINT "♥"
6Ø FOR X=1 TO 52
7Ø XX=INT(RND(1)*52)+1
8Ø DD$=D$(X):D$(X)=D$(XX):
   D$(XX)=DD$:IF RND(1)<.5Ø THEN 13Ø
9Ø PRINT "■■■■■":T=RND
   (1)*3+TX
1ØØ PRINT TAB(T)"■┌‾‾‾‾‾‾‾
   ┐": FOR ZZ=1 TO 9
11Ø PRINT TAB(T)"■|■▒▒▒▒▒
   ■ |":NEXT ZZ
12Ø PRINT TAB(T)"└_______┘"
13Ø NEXT X
14Ø Z=Ø
15Ø PRINT "♥■■■■■■■■■
   ■■■"
16Ø FOR ZZ=1 TO 1Ø:PRINT
   "■■";E1$:NEXT ZZ
17Ø PRINT "■■■■■";:PRINT
   "■■"E2$;E2$
18Ø Z=Z+1:NU=NU+1:IFZ>52 THEN
   Z=1
19Ø DD$=LEFT$(D$(Z),1)
2ØØ D1$=RIGHT$(D$(Z),1):IF D1$="Ø"
   THEN D1$="1Ø":D2$="1Ø": GOTO
   22Ø
21Ø D2$="□"+D1$:D1$=D1$
   +"□"
22Ø FOR D=1 TO 5ØØ:NEXT D:
   PRINT"■CARD■□□□■■■■■";Z
24Ø PRINT "■■■■■■■■■■"
   TAB(T)"■┌‾‾‾‾‾‾‾‾┐":
   FOR ZZ=1 TO 9
25Ø PRINT TAB(T)"■| ■□□□□□
   ■ |":NEXT ZZ
26Ø PRINT TAB(T)"■└________┘
   ■":IF DD$="♦" OR DD$="♥"
   THEN PRINT"■"
27Ø PRINT "■■■■■■■■■■"
   TAB(T);"■";D1$;
   "■■■■■■■■■■";D2$
28Ø FOR ZZ=1 TO 13:IF MID$
   (CC$,ZZ,1)=RIGHT$(D$(Z),1) THEN
   JJ=ZZ
29Ø NEXT ZZ:PRINT "■■■
   ■■■■■■■"TAB(T);"■";
3ØØ ON JJ GOSUB 7ØØ,52Ø,53Ø,54Ø,55Ø,
   56Ø,57Ø,58Ø,59Ø,61Ø,65Ø,67Ø,63Ø
51Ø GOTO 18Ø
52Ø PRINT"■■■"DD$
   "■■■■■"DD$:RETURN
53Ø PRINT"■■■"DD$"■■■"
   DD$"■■■"DD$:RETURN
54Ø PRINT"■■"DD$"■"DD$
   "■■■■■■■■"DD$
   "■"DD$:RETURN
55Ø PRINT"■■"DD$"■"DD$
   "■■■■■■"DD$"■■■■■■"
   DD$"■"DD$:RETURN
56Ø PRINT"■■"DD$"■"DD$
   "■■■■■■■"DD$"■"
   DD$"■■■■■■■"DD$
   "■"DD$:RETURN
57Ø PRINT"■■"DD$"■"DD$
   "■■■■"DD$"■■■■"
   DD$"■"DD$"■■■■■■■"
   DD$"■"DD$:RETURN
58Ø PRINT"■■"DD$"■"DD$
```



```
"███▣"DD$"███▣"
DD$"▣"DD$"███▣"
DD$"███▣"DD$"▣"
DD$:RETURN
59Ø PRINT"▣▣"DD$"▣"DD$
"█████▣"DD$"▣"DD$"██
██▣"DD$"███▣"DD$"▣"
DD$"█████▣"DD$"▣"DD$
6ØØ RETURN
61Ø PRINT "▣"DD$"▣"DD$
"███▣"DD$"███▣"DD$
"▣"DD$"█████▣▣"
DD$"▣"DD$"███▣"
DD$"███▣";
62Ø PRINTDD$"▣"DD$:RETURN
63Ø PRINT"▣◥◥◥◥◥▃▣██
███████▒▒▒▒▒▣██████
███▒●◲●▒▣█████████
▒□◳□▒▣███████████▒
◳▤◰▒▣█████████
◳▤▤▤◰";
64Ø PRINT"▣█████████◲▣□"
DD$"□▃◱":RETURN
65Ø PRINT"□□▃□□▣██████
███▃▒▒▒▃▣████████
▒●□●▒▣██████████▒□
□□▒▣███████████◫◳▤◰
◫▣█████████◳▤▤▤◰";
66Ø PRINT"▣█████████□▣□"
DD$"□▃":RETURN
67Ø PRINT"□▣▚□▃▚□▣██
██████▣▚▚▃□▚▚▣██
██████▣▚▃●□●";
68Ø PRINT"▚▣█████████▣▚
▃□○□▚▣█████████▣▚
▃◳▤◰▚▣█████████▣▚
▃◱□◪▚";
69Ø PRINT"▣█████████□▣◤"
DD$"◥▃":RETURN
7ØØ PRINT"▣▣"DD$"███
▣▃▒▃▣██████▒□▒▣██
███▒▃▒▣█████▒□▒▣
█████▒□▒▣████"DD$:
RETURN
```

The two POKEs in Line 1Ø of the 64 program set the background and border colours, whilst the single POKE in the Vic 20 program looks after the screen colour. MU is the chips that the player will need later on in the game, JM is used for displaying the cards on the screen, and TX is used to centre the cards during the shuffling display that you'll see when you RUN the program. LE is length of the Commodore's screen minus one.

Line 2Ø is concerned with the graphics used as a background for the cards—they are displayed on the screen later on in the program by Lines 16Ø and 17Ø.

In Line 3Ø a string array is DIMensioned, and two strings are defined. D$ will hold the 52 cards, C$ contains the four suit symbols and CC$ contains the numbers and letters used in the corners of the card graphics.

The pack of cards is set up in Lines 4Ø and 5Ø. Now that the pack has been created in order, it will have to be shuffled—Lines 6Ø to

13Ø take care of this. Lines 1ØØ to 12Ø show a screen display of the cards being shuffled.

Once the cards have been put in a random order by the shuffling routine, they can be dealt on to the screen. Line 18Ø increments the card number within the pack—Z—and the number of cards so far dealt—NU. The line also sends the card counter back to the beginning of the pack once the last card has been reached—hence the 52 in this line.

Lines 19Ø to 3ØØ work out which card is at the top of the pack, and then displays it. The value and suit of the card are worked out in Lines 19Ø to 21Ø. There's a pause before the card number is displayed by Line 22Ø. The outline of the card is drawn starting seven lines down on the screen.

You now have a white rectangle—a naked card—which needs dressing up with suit symbols and numbers or letters. Line 27Ø looks after the numbers and letters—a character from ace to king is displayed in the top left-hand corner of the card, and in the bottom right.

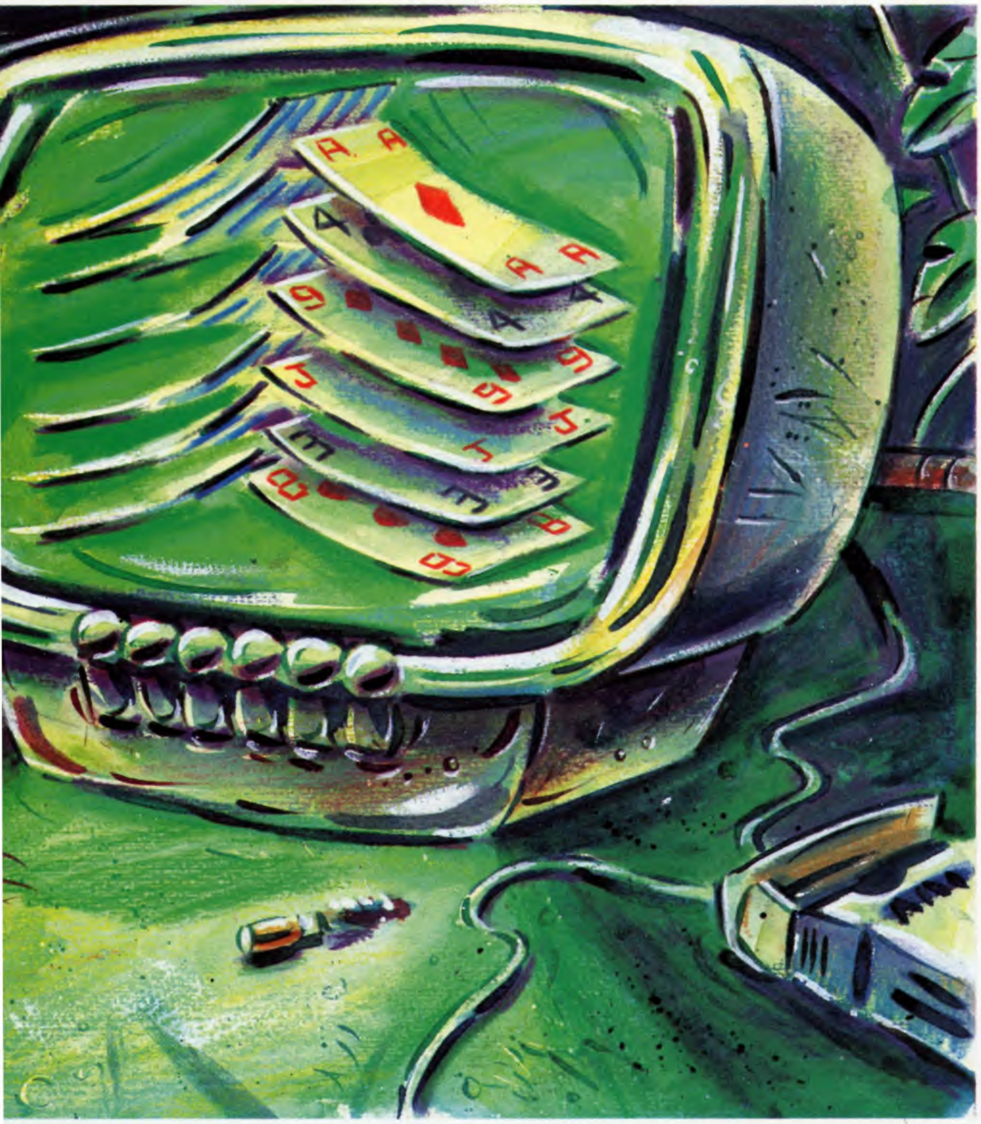
In order that the correct graphics can be displayed on the screen the program has to decide exactly what the card is—is it a 3, an 8 or a Queen? JJ is a number between 1 and 13, representing ace to king, and it is used to pick out the correct line of graphics symbols. These graphics symbols are written as subroutines between Lines 51Ø and 7ØØ. Every time DD$ occurs in these lines, the suit symbol is PRINTed on screen. Lines 51Ø to 6ØØ are the layouts of the spots, and Lines 61Ø to 7ØØ are the heads on the picture cards, which are built up from a combination of the Commodore's block graphics symbols.

The first part of the pontoon program consists of routines which handle the graphics. Press BREAK to clear any old UDGs then enter and RUN these lines to see the cards:

```
20 MODE1
30 PROCSCREEN
40 PRINT"HOLD ON WHILE I OPEN THE
   CARDS"
50 VDU23,224,8,28,62,127,62,28,8,0
60 VDU23,225,8,28,62,127,127,28,62,0
70 VDU23,226,54,127,127,127,62,28,8,0
80 VDU23,227,8,28,28,107,127,107,8,28
90 VDU23,228,0,76,82,82,82,82,76,0
100 VDU23,229,0,60,100,74,67,100,60,0
110 VDU23,230,20,30,36,84,198,38,54,29
120 VDU23,231,73,107,127,65,85,73,34,28
130 DIM C$(52),V(52)
140 D$="A23456789"+CHR$(228)
    +CHR$(229)+CHR$(230)+CHR$(231)
150 L$="JQK"
160 S$=CHR$(224)+CHR$(225)
    +CHR$(226)+CHR$(227)
170 FOR P=0 TO 3
180 FOR T=0 TO 12
190 C$(P*13+T+1)=MID$
    (D$,T+1,1)
```



```
200 V(P*13+T+1)=T+1:IF T>9 THEN
    V(P*13+T+1)=10
210 IF T=0 THEN V(P*13+T+1)=11
220 C$(P*13+T+1)=C$(P*13
    +T+1)+MID$(S$,P+1,1)
230 NEXT
240 NEXT
260 PROCSHUFFLE
270 FOR T=1 TO 52
280 PROCCARD (500,700,T)
290 G=GET
300 NEXT
310 GOTO 260
560 DEF PROCSHUFFLE
570 PRINT"I'M SHUFFLING THE CARDS
    NOW"
580 CN=1
590 X=13:Y=RND(52)
600 FOR P=1 TO 250
610 X=Y-1:IF X=0 THEN X=52
620 Y=RND(52)
630 T$=C$(X):C$(X)=C$(Y):
    C$(Y)=T$
640 T=V(X):V(X)=V(Y):V(Y)=T
650 NEXT
660 ENDPROC
670 DEF PROCCARD(X,Y,Z)
680 LOCAL N
690 VDU5
700 GCOL0,3
710 MOVEX,Y:MOVE X,Y+304:
    PLOT85,X+152,Y:PLOT85,
    X+152,Y+304
720 S=ASC(RIGHT$(C$(Z),1))
730 IF S=225 OR S=227 THEN GCOL0,2
    ELSE GCOL0,1
740 A$=LEFT$(C$(Z),1)
750 MOVEX+10,Y+290:PRINTA$
760 MOVEX+115,Y+40:PRINTA$
770 N=VAL(A$)
780 IF (N MOD 2=1 AND N<>7) THEN
    MOVEX+60,Y+166: VDU S
790 IF N=2 OR N=3 THEN MOVEX+60,
    Y+270:VDU S:MOVEX+60,Y+64:
    VDU S:GOTO 900
800 IF N=4 OR N=5 THEN NS=2
810 IF N=6 OR N=7 OR N=8 THEN
    NS=3
820 IF N=9 OR ASC(A$)=228 THEN
    NS=4:GOTO 840
830 IF N=0 THEN MOVEX+42,Y+286:
    PRINTMID$(L$,ASC(A$)-830,1):
    MOVEX+80,Y+40:PRINTMID$
    (L$,ASC(A$)-830,1):MOVEX+60,
    Y+166:VDU S:GOTO 900
840 FOR T2=0 TO NS-1
850 MOVEX+20,Y+80+170/(NS-1)
    *T2:VDUS
860 MOVEX+100,Y+80+170/(NS-1)
    *T2:VDUS
870 NEXT
880 IF N=7 THEN MOVEX+60,Y+205:
    VDU S
890 IF N=8 OR ASC(A$)=228 THEN
    MOVEX+60,Y+205:VDU S:
    MOVEX+60,Y+120:VDU S
900 VDU4
920 ENDPROC
1400 DEF PROCSCREEN
1410 VDU28,0,18,39,14
1420 VDU19,0,2,0,0,0:VDU19,2,0,0,0,0
1430 ENDPROC
```



The cards are drawn on a MODE1 screen so Line 20 sees to this. Line 30 calls PROCSCREEN which is located at Lines 1400 to 1430. Line 1410 defines a text window in the centre of the screen, and the colours are set up by Line 1420.

The player is told HOLD ON WHILE I OPEN THE CARDS. Lines 50 to 120 set up graphics for the suit symbols along with the figure 10—set up as one character for ease of display later on—and the three heads used in the corners of the J, Q, K cards.

Two arrays are DIMensioned in Line 130—C$ will hold the suits and the numbers and letters used in the cards, and V will hold the values of each of the 52 cards. D$ holds all the numbers and the heads, whilst L$ holds J, Q, K. S$ holds the four suit symbols.

Next, C$ and V are filled using the two FOR ... NEXT loops in Lines 170 and 180 and Lines 230 and 240. In the case of C$, Line 190 picks out the elements from D$ and puts them in the correct element of the array. The remainder of the array is filled from S$ by Line 220. The values of the cards are fed into array V by Line 200—all cards above nine have the value ten, so the last part of the line looks after that. Finally, the ace can take two values, 11 as well as one, so Line 210 checks if the card is an ace and puts the value 11 into V.

Line 260 calls PROCSHUFFLE, found at Lines 560 to 660. After the player has been told I'M SHUFFLING THE CARDS NOW, by

Line 57Ø, the card number CN is set to one by Line 58Ø. PROCSHUFFLE works by picking two cards and then exchanging them. The FOR . . . NEXT loop between Lines 6ØØ to 65Ø shuffles the cards 250 times. Line 59Ø starts the process by picking card 13 and a random card and swapping them. The shuffle proceeds—Lines 61Ø to 64Ø—by subtracting one from the last random position and swapping it with a new random card. And so on for 250 swaps.

Line 27Ø sets up a FOR . . . NEXT loop which ensures that 52 cards are dealt. Line 28Ø calls PROCCARD which displays the card. The two numbers ensure that the cards are displayed at the correct position—X and Y are the coordinates of the bottom left hand corner of the card. A variable, N, is defined as LOCAL in Line 68Ø. VDU5 in Line 69Ø allows text to be printed at the graphics cursor.

The white background is displayed by Line 71Ø. Line 72Ø enables you to find out if the card is a red or a black suit. Line 73Ø tests the value of S, and sets either red or black—GCOLØ,2 is black, and GCOLØ,1 is red. With the colour set, Lines 75Ø and 76Ø place the characters at the corners of the cards.

The spots are displayed by Lines 77Ø to 89Ø. Line 77Ø takes the number of the card in A$, and then Lines 78Ø to 89Ø look at N and display the spots in the correct positions.

Line 9ØØ switches back to the text cursor.

Line 29Ø makes the program pause until a key is pressed, when it displays a new card. After 52 cards have been dealt Line 31Ø repeats the shuffling and dealing loop.

The section of program for the Dragon and Tandy machines which handles the card graphics is as follows. Type it in and RUN it and you'll be dealt 52 cards, one at a time:

```
10 PMODE3,1
20 CLS:PRINT@228,"I'M SHUFFLING THE
   CARDS"
30 DIMNU$(13):FOR K=1 TO 13:READ
   NU$(K):NEXT
40 DATA BD2S4U5ER5FD2NL5D3,RDLDR,
   RDNLDL,DRDU2,NRDRDL,D2RUL,
   RS8DGD,ND2RDNLDL,NDRDNLD,
   D2S16BRS12NU2RU2L,S4BD5F2R
   2U6L3R4,S4NR5D6R3NH2NFR2U6,
   DNDS8RS12NEF
50 FORK=1536TO43*32+1536 STEP 32
60 READA,B:POKEK,A:POKEK+1,B:NEXT
70 DATA 60,224,184,184,238,236,187,
   184,238,236,187,184,46,224,59,
   176,14,192,11,128,2,0
80 DATA 2,0,3,0,11,128,14,192,59,176,
   238,236,59,176,14,192,11,128,3,0,3,0
90 DATA 1,0,6,64,9,128,38,96,25,144,
   102,100,153,152,102,100,153,
   152,34,32,1,0
100 DATA 2,0,9,128,6,64,9,128,34,32,
   153,152,102,101,153,152,102,
   101,17,16,2,0
110 DIM C(3),D(3),H(3),S(3),SQ(61)
120 GET(0,0)-(13,10),H,G
130 GET(0,11)-(13,21),D,G
140 GET(0,22)-(13,32),S,G
150 GET(0,33)-(13,43),C,G
160 FORK=0 TO 51:SQ(K)=K:NEXT
170 SCREEN 1,1
180 GOSUB 1500: N=0
190 PCLS 6: FOR CX=6 TO 200 STEP 50:
   FOR CY=11 TO 108 STEP 97
200 GOSUB 1000: GOSUB 2000: FOR J=1
   TO 500: NEXT J,CY,CX
210 FOR J=1 TO 1000: NEXT: GOTO 190
1000 ST=INT(SQ(N)/13)+1:NM
   =SQ(N)-13*ST+14:N=N+1:
   IF N>51 THEN N=0
1010 RETURN
1500 FORX=52 TO 2 STEP-1
1510 Q=RND(X)
1520 T=SQ(X-1):SQ(X-1)=SQ(Q-1):
   SQ(Q-1)=T
1530 NEXT
1540 FORX=0TO9:SQ(X+52)=SQ(X):
   NEXT
1550 RETURN
2000 LINE(CX,CY)-(CX+44,CY+72),
   PRESET,BF
2010 S$=NU$(NM)
2020 IF ST>2 THEN COLOR7 ELSE COLOR8
2030 DRAW"S12BM"+STR$(CX+3)+","
   +STR$(CY+2)+S$
2040 DRAW"S12BM"+STR$(CX+35)+","
   +STR$(CY+64)+S$
2050 IF (NM/2<>INT(NM/2)AND
   NM<>7)ORNM>10 THEN PX=
   CX+16:PY=CY+31:GOSUB2500
2060 IFNM=2 OR NM=3 OR NM=10 OR
   NM=8 THENPX=CX+16:PY=CY+19:
   GOSUB2500:PY=PY+24:GOSUB2500
2070 IF NM=7 THEN PX=CX+16:
   PY=CY+39:GOSUB2500
2080 IF NM<4 OR NM>10 THEN 2140
2090 IF (NM=10 OR NM=8)THEN
   NS=INT((NM-1)/2) ELSE
   NS=INT(NM/2)
2100 FOR J=0 TO NS-1
2110 PX=CX+3:PY=CY+12+J*38/
   (NS-1):GOSUB2500
2120 PX=CX+30:GOSUB2500
2130 NEXT
2140 RETURN
2500 ON ST GOTO 2510,2520,2530,2540
2510 PUT(PX,PY)-(PX+13,PY+10),
   H,OR:RETURN
2520 PUT(PX,PY)-(PX+13,PY+10),
   D,OR:RETURN
2530 PUT(PX,PY)-(PX+13,PY+10),
   C,OR:RETURN
2540 PUT(PX,PY)-(PX+13,PY+10),
   S,OR:RETURN
```

A four-colour mode has been chosen so that the suit symbols can be displayed in blue and red. Line 1Ø, then, sets PMODE3. Line 2Ø tells the player that the cards are being shuffled.

As the cards are drawn on the high resolution screen, no characters are available via the keyboard. The Ace, King, Queen and Jack symbols, and the numerals from two to ten are DRAWn from the DATA in Line 4Ø. The DRAWing instructions are READ into array NU$ by Line 3Ø.

The symbols for hearts, diamonds, spades and clubs are POKEd on to the screen from the DATA in Lines 7Ø to 9Ø. As soon as the symbols have been displayed, GET is used to place the symbols in memory—Lines 12Ø to 15Ø. The arrays for the symbols have been DIMensioned in Line 11Ø, along with array SQ which is used for shuffling the cards. Line 16Ø fills the first 52 elements of SQ with numbers 0 to 51.

Line 18Ø calls the shuffling subroutine—Lines 15ØØ to 155Ø—which orders the cards randomly. N is set equal to zero in Line 18Ø so that the first card in SQ is being looked at.

Line 19Ø colours the screen cyan and sets up two FOR . . . NEXT loops using variables CX and CY, which will be used later for positioning symbols on the cards.

Line 2ØØ calls two subroutines. Firstly, Lines 1ØØØ and 1Ø1Ø calculate the suit—ST—and the number—NM—of the card in SQ. Lines 2ØØØ to 214Ø calculate the positions that the symbols will have to be PUT on the screen to represent that card.

The subroutine looks very complicated, but with the help of a pack of cards you should be able to understand what is happening. There are several recurring patterns which make up the cards—the lower cards just use one pattern, and the higher cards use a combination of two or more. The subroutine checks which of the patterns are needed for the displayed card.

But before the spots are worked out, the numbers or letters are DRAWn in the two opposite corners of the card by Lines 2Ø3Ø and 2Ø4Ø using the information in array NU$.

The positions of the spots are worked out by checking the value of the card in Lines 2Ø5Ø to 2Ø9Ø. PX and PY are the spot coordinates and are worked out from the values of CX and CY.

Once PX and PY have been calculated, the subroutine at Line 25ØØ is called. This routine PUTs symbols of the correct suit on the screen.

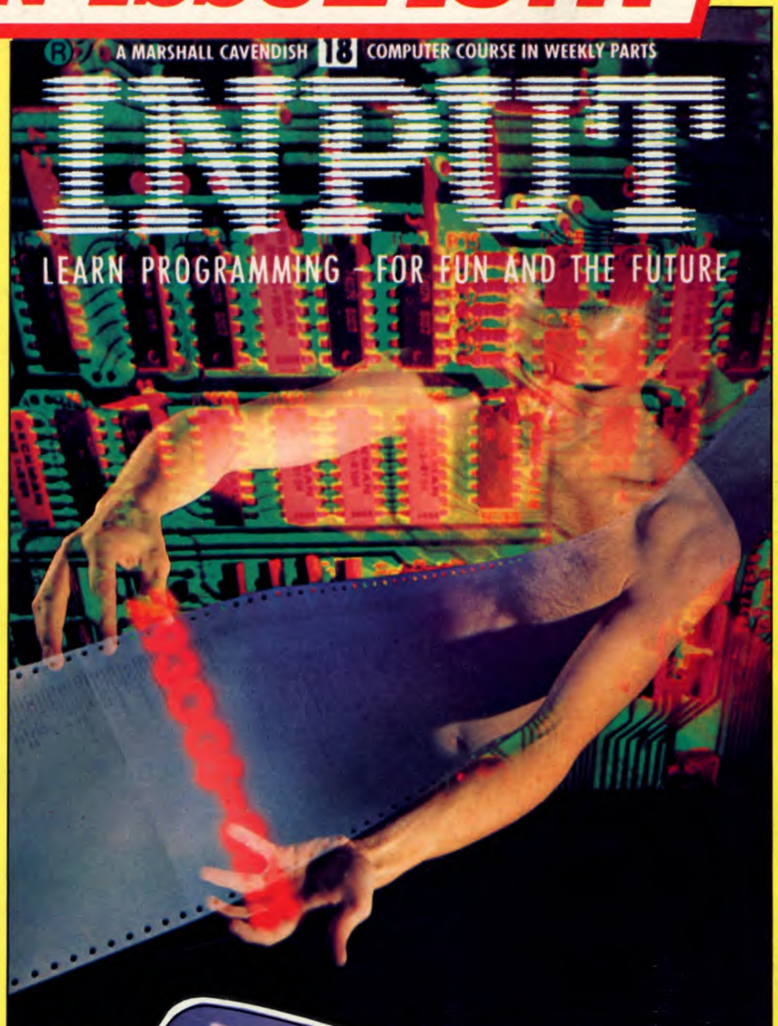
Line 21Ø inserts a pause before the program loops back to Line 19Ø where the screen is cleared and another card is displayed.

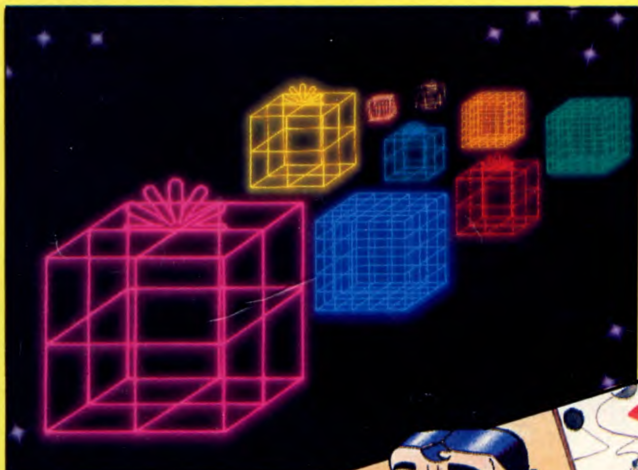
CUMULATIVE INDEX

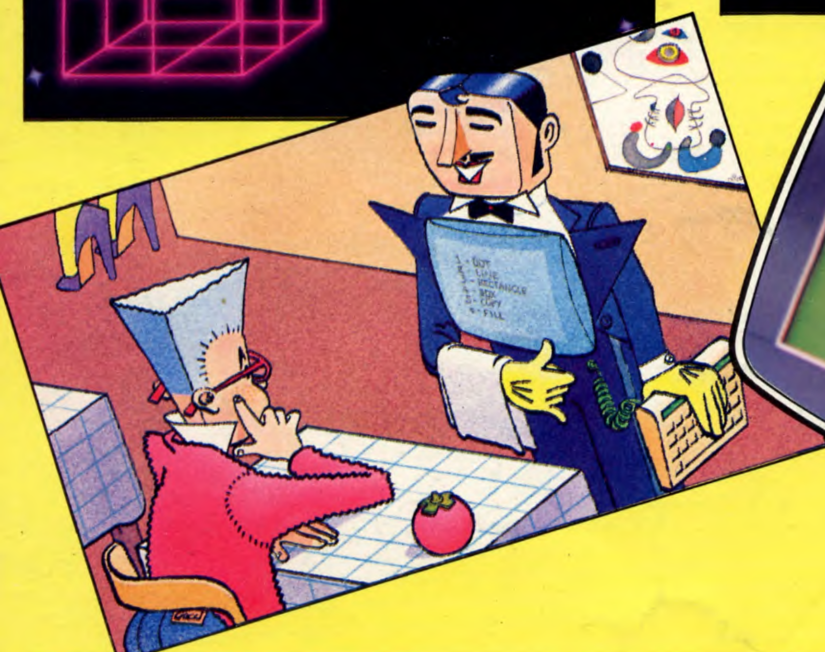
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 18...

- ❑ *Type in the COMPUTER AIDED DESIGN program and create detailed drawings with easy fingertip control*
- ❑ *With your pack of cards ready and shuffled, it's time for THE PLAYER'S TURN AT PONTOON*
- ❑ *Now you have learned to build flat shapes, it's time to start creating WIREFRAMES IN 3-D*
- ❑ *Discover the advantages of WORD-PROCESSING—the sophisticated office tool that's finding its way into the home*
- ❑ *And for ACORN users, there's a MACHINE CODE PACKER*









ASK YOUR NEWSAGENT FOR INPUT